MASTER EN SCIENCES INFORMATIQUES

Construction d'arbres de décision adaptée aux attributs quantitatifs et à l'asymétrie

Elias, Arnaud

Award date:
2011

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Faculté d'Informatique
Année académique 2010 - 2011

Construction d'arbres de décision adaptée aux attributs quantitatifs et à l'asymétrie

Arnaud Elias

Mémoire présenté en vue de l'obtention du grade de Master en sciences informatiques

Résumé

En data mining, la classification supervisée permet, après une phase d'apprentissage d'un modèle à partir d'un jeu de données, d'appliquer ce modèle à de nouveaux jeux de données de structure identique au premier.
Dans ce travail, nous nous sommes penchés sur deux spécificités du jeu de données qui peuvent complexifier la tâche d'apprentissage et donner lieu à de faibles performances de classification : l'asymétrie des données et les attributs indépendants quantitatifs.
Dans le but d'améliorer les performances de classification sur ce type de données, nous avons construit un classifieur utilisant plusieurs techniques connues pour leur adéquation par rapport à ces deux spécificités. Notre solution est basée sur un algorithme de création d'arbres de décision et possède plusieurs instances, chacune ayant des propriétés particulières.
Afin d'être en mesure de qualifier notre solution, nous évaluons et comparons les résultats obtenus par les différentes instances sur plusieurs jeux de données possédant ces spécificités.

Mots-clés : data mining, jeux de données asymétriques, arbres de décision, règles, classification, discrétisation, attributs quantitatifs, classe minoritaire

Abstract

In data mining, supervised classification learns models from datasets and allows the application of these model on new datasets with the same structure. In this work, we take an interest in two characteristics of the datasets that make learning task more complex and cause poor classification performances : unbalanced datasets and quantitative attributes.
In order to improve classification performances on this type of data, we developed a classifer using several techniques known for matching these two characteristics. Our solution is based on a decision tree creation algorithm and has several instances, each having particular specificities.
In order to assess our solution, we evaluate and compare the results of its different instances on several datasets with such characteristics.

Keywords: data mining, unbalanced datasets, decision trees, rules, classification, discretization, quantitative attributes, minority class

Remerciements

Je tiens tout d'abord à remercier mon promoteur, Monsieur Vanhoof qui a accepté de me superviser durant toute la durée de mon travail malgré ses nombreuses autres obligations. Un grand merci également à l'un de ses assistants, Monsieur Marcozzi, pour sa relecture consciencieuse et ses conseils avisés.

Il me tient également à coeur de remercier l'ensemble des personnes qui ont permis le bon déroulement de mon stage à l'étranger. Merci donc à Messieurs Ruiz et Garcia Torres, professeurs à l'Université Pablo de Olavide de Séville qui, par leur sens de l'accueil et leur grande disponibilité ont grandement facilité mon intégration. Merci aussi à Messieurs Asencio Cortés et Márquez Chamorro, chercheurs à l'université Pablo de Olavide d'avoir rendu mes journées de stage aussi agréables.

Merci également à Kathleen et à mes parents pour leur aide et leur soutien indéfectible.

Enfin, je remercie les membres du jury pour le temps qu'ils consacrent à ce travail.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

1 Introduction

En data mining, la *classification supervisée* se fait en deux phases. La première phase est la phase inductive durant laquelle nous construisons un modèle à partir d'un jeu de données particulier. Après cette étape de généralisation du modèle, vient la phase déductive lors de laquelle nous appliquons le modèle construit à un nouveau jeu de données. Ce type de classification a pour but d'affecter à chaque instance du nouveau jeu de données un type particulier : une *classe*.

La classification supervisée permet donc de prédire la classe des instances de jeux de données divers et variés. Cette technique est largement répandue et utilisée dans des domaines tels que l'informatique, afin de prédire si un e-mail reçu est un spam ou non ; la médecine, lorsqu'il s'agit de détecter la probabilité pour un patient particulier de contracter une maladie ou encore dans le milieu bancaire, afin de déceler les transactions frauduleuses.
Suivant les spécificités du jeu de données, nous percevons que la classification peut être plus ou moins complexe.

Deux des trois exemples précédents ont une spécificité qui rend l'apprentissage à partir de ceux-ci plus complexe. En effet, pour les exemples de détection de maladies et de transactions frauduleuses, les jeux de données contiennent une classe minoritaire, c'est-à-dire une classe faiblement représentée dans le jeu

de données. Dans ces deux types de jeux de données, le nombre de cas positifs, c'est-à-dire le nombre de patients malades ou le nombre de transactions frauduleuses est faible par rapport aux cas négatifs, à savoir les patients sains et les transactions électroniques en règle.

Construire un modèle de bonne qualité lorsque l'on est face à des jeux de données de ce type n'est pas une tâche aisée car il est difficile de généraliser à partir d'un faible nombre d'instances d'une classe. De plus, lors de l'étape inductive, de nombreuses techniques favoriseront la classe majoritaire des jeux de données, la classe minoritaire sera donc quasiment absente alors que c'est celle qu'il faut généralement détecter correctement.

Le présent travail se focalise sur les jeux de données présentant cette spécificité, c'est-à-dire les jeux de données *asymétriques*.

La deuxième caractéristique des jeux de données étudiés dans le cadre de ce travail est qu'ils sont composés d'attributs indépendants *quantitatifs*. Lorsqu'une technique classique de classification supervisée doit faire face à des attributs indépendants quantitatifs, elle *discrétise* ceux-ci. Les méthodes de discrétisation utilisées sont généralement peu adaptées aux jeux de données asymétriques. Afin de tenter d'améliorer les performances de classification sur les jeux de données présentant ces deux spécificités, nous avons conçu (et implémenté) un *classifieur* utilisant des techniques spécialement adaptées à ceux-ci.

Notre classifieur est un algorithme de création d'*arbres de décision* qui possède deux variantes, l'une étant entièrement adaptée aux jeux de données ci-dessous et l'autre partiellement. La technique la plus adaptée à ce type de jeux de données utilise la discrétisation locale suivant une mesure d'impureté connue pour donner de bonnes performances face aux jeux de données asymétriques. Elle emploie également une méthode de post-élagage au moyen du test exact de Fisher, méthode connue pour donner de bons résultats par rapport aux jeux de données asymétriques.

Ce travail s'articule en trois grands chapitres. Dans le chapitre 2, nous expliquerons les bases nécessaires afin de disposer d'un bagage suffisant en vue de comprendre le problème ainsi que la solution proposée à celui-ci. Nous détaillerons le data mining, discipline dans laquelle évolue notre travail ainsi que la classification supervisée avant d'exposer une méthode particulière de classification : les arbres de décision. Nous présenterons également plusieurs mesures

permettant de déterminer la qualité des classifieurs produits. Les deux dernières sections de ce chapitre correspondent respectivement à une présentation générale des deux spécificités relatives aux jeux de données étudiés et présentent quelques méthodes permettant de faire face à ces spécificités.

La troisième partie de ce mémoire détaillera le problème ainsi que la solution proposée. Nous exposerons tout d'abord l'algorithme de base avant de raffiner celui-ci au moyen de deux instances. Nous détaillerons également les choix qui nous ont orientés vers ce type de solution.

La troisième partie contient les résultats de l'expérimentation ainsi qu'une analyse critique de ceux-ci.

Enfin, nous conclurons ce travail en récapitulant l'ensemble des apports, en résumant les avantages et inconvénients de la technique et en donnant des pistes afin de continuer et d'améliorer le présent ouvrage.

2

Etat de l'art

Dans ce chapitre, nous présentons les principaux concepts et méthodes permettant d'offrir au lecteur les connaissances suffisantes afin qu'il puisse comprendre aisément la solution proposée au problème posé.

Pour cela, nous détaillerons ceux-ci en six parties. Nous commencerons par expliquer la discipline dans laquelle évolue notre travail, à savoir le **data mining** (Section 2.1).
Ensuite et après avoir expliqué plus en détails ce que nous entendons par **classification** (Section 2.2), nous nous concentrerons sur une méthode permettant de classer des instances d'un jeu de données : les **arbres de décision** (Section 2.3).
Par après, nous ferons un parallèle entre les arbres de décision et les **règles**, celles-ci permettant de représenter de manière plus formelle les sous-ensembles (les feuilles de l'arbre) issus de l'arbre de décision (Section 2.4).
La section suivante permettra d'apporter des informations supplémentaires quant à la **qualité** des règles produites (Section 2.5).
Ensuite, les méthodes utilisées dans notre solution pour **discrétiser** les attributs quantitatifs d'un jeu de données seront exposées (Section 2.6).
Enfin, nous expliquerons les transformations à effectuer dans les concepts et méthodes présentés afin d'appréhender de manière plus efficace les jeux de données **asymétriques** (Section 2.7).

2.1 Data Mining

Dans cette partie, nous présentons les concepts fondamentaux du data mining.

Le **data mining** est une discipline qui a pour objectif d'extraire du savoir, des connaissances, des patterns et ainsi obtenir une plus-value à partir de grandes quantités de données [WFH11].
Cette discipline est utile dans des domaines éclectiques tels qu'en médecine, pour détecter une maladie au début de son occurrence [DWK05] ou pour quantifier le risque de développer un problème physique, comme par exemple une anomalie cardiaque [GLK03]. Le data mining peut également aider la communauté scientifique étudiant la bioinformatique à approfondir ses connaissances sur le génome humain [hum00].
En marketing, cette discipline permet par exemple de détecter un groupe de consommateurs cible pour une prochaine campagne de publicité [LCGF04].

Comme les exemples précédents le montrent, les techniques appartenant au data mining peuvent être appliquées sur des données provenant de domaines variés. Néanmoins, ces données, nommées des **jeux de données** gardent sensiblement la même structure, similaire à celle du jeu de données de la Table 2.1. Cet exemple est composé de plusieurs individus qui possèdent chacun des caractéristiques propres. Celui-ci peut être vu comme une partie d'une base de données d'une organisation qui fournit une aide financière suivant l'approbation du dossier que l'individu a fourni. Le dossier de chaque individu contient les caractéristiques principales de celui-ci. L'approbation ou non du dossier se fait à partir des caractéristiques propres de l'individu, comme son diplôme, son statut civil, son sexe et s'il a des enfants. L'identifiant n'est pas un attribut du jeu de données mais a pour unique rôle de pouvoir reconnaître rapidement une instance particulière.

Un jeu de données typique S peut être défini plus formellement par une matrice S_M :

$$S_M = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

Identifiant	Diplôme	Etat civil	Sexe	Enfants	Dossier approuvé ?
1	primaire	célibataire	homme	non	non
2	primaire	célibataire	homme	oui	non
3	universitaire	marié	homme	non	oui
4	universitaire	divorcé	femme	non	oui
5	universitaire	marié	femme	oui	oui
6	secondaire	marié	homme	non	non
7	universitaire	célibataire	femme	non	oui
8	secondaire	divorcé	femme	non	oui
9	secondaire	célibataire	femme	oui	oui
10	universitaire	marié	homme	oui	oui
11	universitaire	marié	femme	non	oui
12	secondaire	divorcé	homme	oui	non
13	universitaire	divorcé	femme	oui	non
14	secondaire	divorcé	homme	non	oui
15	primaire	divorcé	homme	non	non

TABLE 2.1 – Exemple de jeu de données

Cette matrice comprend m lignes et n colonnes. Chaque ligne i de la matrice S_M : $\begin{pmatrix} a_{i,1} & a_{i,2} & \cdots & a_{i,n} \end{pmatrix}$ est une **instance**. Chaque colonne j de la matrice S_M : $\begin{pmatrix} a_{i,2} \\ \cdots \\ a_{i,n} \end{pmatrix}$ est un **attribut**. Les attributs vont servir à caractériser les instances. Un élément $a_{i,j}$ de la matrice est la valeur que possède l'instance i pour l'attribut j. Les attributs peuvent être de différents types [1] [min, WFH11, YWW10] :

- Les attributs **qualitatifs** sont des données catégoriales, c'est-à-dire des attributs qui peuvent être insérés dans différentes catégories. Un ordre peut parfois être appliqué sur ceux-ci, mais pas d'opérations arithmétiques. Les deux principaux types d'attributs qualitatifs sont les attributs qualitatifs **ordinaux** et **nominaux**.

  - **ordinal** : un attribut ordinal est un attribut qui comprend un nombre fini de valeurs et sur lesquelles nous pouvons appliquer une relation d'ordre, mais aucune opération arithmétique. L'attribut "Diplôme" du jeu de données de la Table 2.1 est un attribut de type ordinal car il est possible d'appliquer une relation d'ordre de type $primaire < secondaire < universitaire$.

  - **nominal** : un attribut nominal est un attribut qui comprend un nombre fini de valeurs et sur lesquelles aucune relation d'ordre ni opération

1. Seuls les principaux types sont évoqués ici, une liste plus détaillée est disponible dans [WFH11].

arithmétique ne peut être effectuée. L'attribut "Etat civil" du jeu de données de la Table 2.1 est un attribut nominal.

- Les attributs **quantitatifs** sont quant à eux exclusivement numériques. Un ordre peut-être appliqué sur ceux-ci, ainsi que des opérations arithmétiques. Les attributs quantitatifs peuvent être divisés en deux groupes, **discrets** et **continus**.

  - Les attributs **discrets** ont un nombre fini ou infini dénombrable de valeurs. Un attribut discret d'un jeu de données est par exemple le nombre de propriétés que possède un investisseur immobilier.

  - Les attributs **continus** ont pour valeurs des nombres réels. Un exemple d'attribut continu est la **masse d'un individu**.

Après avoir expliqué la structure générale d'un jeu de données typique du data mining, nous pouvons nous concentrer sur les méthodes de data mining, en nous attardant sur la classification.

Les deux principales approches du data mining sont l'**induction descriptive** et **l'induction prédictive**.
Avant d'expliquer plus précisément celles-ci, nous distinguons ces deux types au moyen d'exemples.

A partir du jeu de données de la Table 2.1, nous pouvons donner un exemple de connaissance pour l'induction descriptive, sous la forme d'une question : "Quel est le pourcentage d'hommes divorcés qui ont des enfants ?". A partir de cette question, nous tentons d'extraire du savoir par rapport aux données.

Afin de pouvoir expliquer l'induction prédictive, nous devons introduire un deuxième jeu de données, représenté par la Table 2.2. Ce jeu de données à la même structure que le jeu de données de la Table 2.1.
Le but de l'induction prédictive est d'apprendre à partir du jeu de données de la Table 2.1 afin d'appliquer la connaissance au jeu de données de la Table 2.2. Cette application permettra d'assigner une valeur d'attribut cible "Dossier approuvé" à chaque instance du jeu de données de la Table 2.2. Par exemple, nous pouvons, à partir d'une technique d'induction prédictive, affirmer que tous les individus mariés verront leur dossier approuvé. Nous appliquerons ensuite cette affirmation au jeu de données de la Table 2.2 afin d'assigner la valeur "oui" à l'attribut "Dossier approuvé" de toutes les instances satisfaisant cette

affirmation.

Identifiant	Diplôme	Etat civil	Sexe	Enfants	Dossier approuvé ?
1	primaire	marié	homme	oui	?
2	universitaire	marié	femme	non	?
3	secondaire	célibataire	homme	oui	?
4	secondaire	divorcé	femme	oui	?
5	universitaire	divorcé	femme	non	?
6	primaire	marié	femme	oui	?
7	universitaire	divorcé	femme	non	?
8	secondaire	divorcé	femme	non	?
9	secondaire	divorcé	femme	oui	?
10	secondaire	divorcé	homme	oui	?

TABLE 2.2 – Exemple de jeu de données sur lequel nous pouvons appliquer les connaissances apprises afin prédire la valeur de “Dossier approuvé”

C’est au niveau de l’**attribut cible** qu’est la différence entre l’induction descriptive et l’induction prédictive. L’induction prédictive essayera d’expliquer les différentes valeurs de l’attribut cible “Dossier approuvé” en fonction des autres attributs tels que le diplôme ou l’état civil alors qu’une technique d’induction descriptive ne prendra pas en compte l’attribut “Dossier approuvé”. L’induction prédictive utilise donc un **attribut cible**, contrairement à l’induction descriptive. Un attribut cible est un **attribut dépendant** dont nous tentons d’expliquer le comportement au moyen de tous les autres attributs (nommés les attributs **indépendants**) du jeu de données.

L’attribut cible d’un jeu de données peut prendre différentes valeurs. Chaque sous-ensemble d’instances du jeu de données ayant une valeur d’attribut cible identique s’appelle une **classe**. La **distribution** du jeu de données est un *n-uplet* contenant les nombres ou pourcentages d’instances appartenant à chaque classe du jeu de données. Ce travail se focalisant sur des jeux de données ne contenant que deux classes, la distribution est donc un couple de la forme (classe positive, classe négative). Pour le jeu de données de la Table 2.1, la distribution est de (9,6).
L’attribut cible de la matrice S_M est la dernière colonne de celle-ci : $\begin{pmatrix} a_{1,n} \\ a_{2,n} \\ \dots \\ a_{m,n} \end{pmatrix}$.
Pour la matrice S_M, l’ensemble des classes est $C = \{C_1, \dots, C_t, \dots, C_q\}$, où chaque $a_{i,n} \in C_t$.
Soit $d(C_t, S)$; la proportion d’instances de classe C_t dans S, la **distribution** du jeu de données S est : $D(S) = (d(C_1, S), d(C_2, S), \dots, d(C_q, S))$.

2.1.1 L'induction descriptive

L'**induction descriptive** a pour principal objectif d'extraire du savoir grâce aux données. Une technique d'induction descriptive tentera de détecter les corrélations entre les instances. Les techniques de description les plus connues sont :

- **les règles d'association** [MIIB] : elles tentent de détecter des structures, des régularités dans les jeux de données en étudiant les relations entre les attributs. Par exemple, cette technique essaye de déterminer quel est le pourcentage de personnes qui, dans un jeu de données d'achats dans un supermarché achètent à la fois du pain et des oeufs [NF].

- **le clustering** [data] : le but du clustering est de diviser une population hétérogène en sous-populations homogènes (en populations d'instances présentant des caractéristiques communes). Un exemple de technique de clustering est le *K-means* [WFH11].

L'exemple du jeu de données de la Table 2.1, après avoir supprimé l'attribut cible "Dossier Approuvé" et sur lequel nous appliquons une méthode de clustering peut être représenté graphiquement par la Figure 2.1.
Dans cette Figure, les instances du jeu de données ont été regroupées en quatre sous-ensembles, après application de la méthode de clustering. Comme expliqué précédemment, les instances ne sont pas groupées suivant leur type (elles ne possèdent pas d'attribut cible) mais celles-ci sont groupées en ensembles composés d'instances similaires [2] mais qui sont différentes des instances des autres ensembles [3]. Pour cet exemple, les quatre clusters créés sont :

cluster rouge : les hommes célibataires

cluster bleu : les femmes célibataires

cluster vert : les hommes mariés ou divorcés

cluster orange : les femmes mariées ou divorcées

2.1.2 L'induction prédictive

L'**induction prédictive** est utilisée dans le but de découvrir du savoir afin de pouvoir classifier, prédire. Cette technique construit un modèle à partir du

2. forte similarité intra-classe [NF]
3. faible similarité inter-classe [NF]

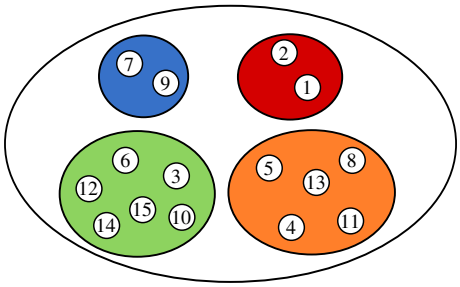

FIGURE 2.1 – Jeu de données de la Table 2.1 après application d'une technique de clustering

jeu de données initial (nommé le jeu de données **d'apprentissage**). Ce modèle est ensuite appliqué à des jeux de données possédant la même structure que le jeu de données d'apprentissage, des jeux de données dont les instances sont dépourvues de valeurs d'attribut cible. C'est au modèle d'assigner à chaque instance de ces jeux de données un attribut cible. Appliquée au jeu de données de la Table 2.1, une technique d'induction prédictive construira un modèle. Ce modèle pourra par la suite, en lui fournissant des jeux de données de la "vie réelle" (contenant des instances non vues auparavant), prédire si un individu verra son dossier approuvé ou non.
Les techniques de prédiction les plus usitées sont :

- **la classification** [cla, datb] : Cette technique va assigner des instances d'un jeu de données à des classes. Pour cela, elle va construire un modèle basé sur un jeu de données d'apprentissage, jeu de données contenant un **attribut cible**. Les arbres de décision sont un exemple de technique de classification. C'est sur ce type d'induction prédictive que repose ce travail.
  Le processus de classification peut être résumé au moyen de la Figure 2.2. Nous apprenons le modèle à partir du jeu de données d'apprentissage et nous l'appliquons ensuite à la vie réelle. Grâce au modèle, nous pouvons prédire la classe de nouvelles instances.

- **la régression** [reg] : Cette technique consiste en la création d'une fonction à partir d'un jeu de données contenant des attributs quantitatifs mettant en relation les attributs indépendants avec l'attribut cible, c'est-à-dire en donnant une valeur d'attribut cible à partir des attributs indépendants. L'attribut cible étant quantitatif, les valeurs de celui-ci sont divisées en "régions" lors de la régression. Ce modèle sera appliqué à des jeux de

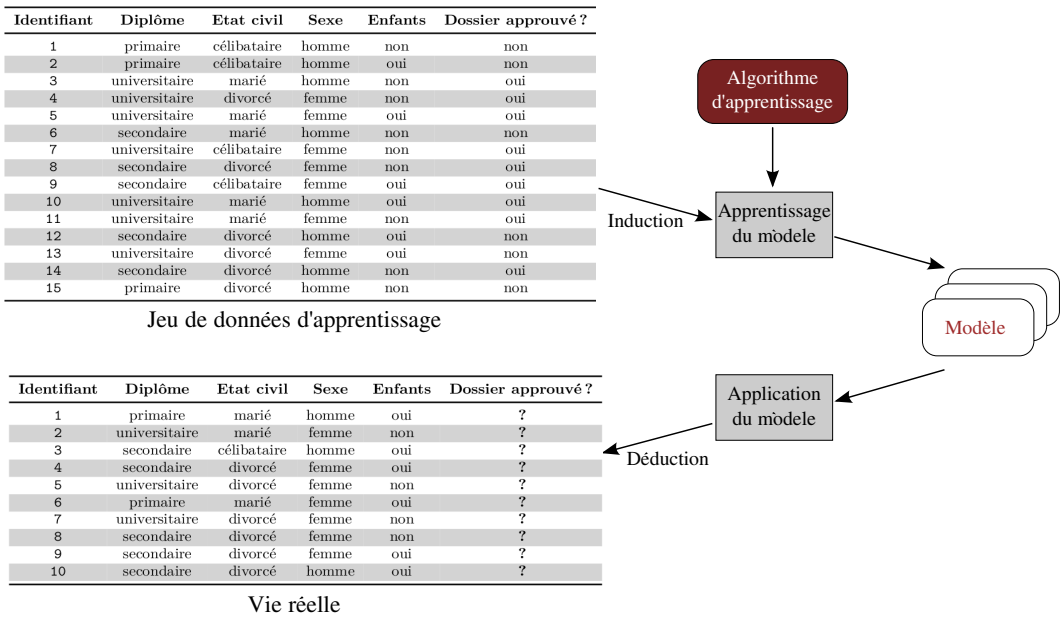

FIGURE 2.2 – Construction et utilisation d'un modèle prédictif [Jin07]

données de test à des fins de prédiction.

Un exemple de classification à partir du jeu de données de la Table 2.1 est représenté par la Figure 2.3. Dans celle-ci, les instances sont classées en deux régions distinctes suivant leur valeur d'attribut cible. La région rouge attribue aux instances de cette région la valeur d'attribut cible "Dossier Approuvé = **non**". La région verte attribue aux instances présentes dans celle-ci la valeur d'attribut cible "Dossier Approuvé = **oui**". Ces régions, créées à partir du jeu de données d'apprentissage, c'est-à-dire le jeu de données de la Table 2.1, vont être appliquées au jeu de données de la Table 2.2 afin de pouvoir assigner une région (une valeur d'attribut cible) à chaque instance de ce jeu de données.

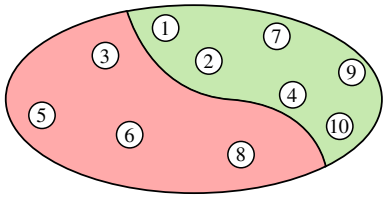

FIGURE 2.3 – Jeu de données de la Table 2.1 après application d'une technique de classification

2.1.3 Problème et solution

Dans le cadre de ce travail, nous avons développé une solution informatique permettant de *classifier* (Section 2.2) des instances venant de jeux de données *asymétriques* (intuitivement, jeu de données où une classe est largement plus représentée que l'autre classe) et dont les attributs indépendants sont *quantitatifs*.
L'algorithme développé pour faire face à ce type de problème est un algorithme créant des arbres de décision (Section 2.3). L'arbre divise progressivement le jeu de données en utilisant une technique de *discrétisation* pour pouvoir faire face aux attributs continus (Section 2.6). L'ensemble des mesures et des techniques utilisées durant le processus de construction de l'arbre de décision tente de ne pas être discriminant par rapport à la classe *minoritaire* (la classe la moins représentée) du jeu de données (Section 2.7). Après la création de l'arbre, les utilisateurs peuvent visualiser les résultats directement à partir de l'arbre de décision ou encore grâce aux *règles* produites (Section 2.4) à partir de celui-ci. Ils peuvent également avoir un compte rendu sur la *qualité* (Section 2.5) des règles produites.

2.2 Classification

Préalablement à l'explication du type de classifieur employé dans ce travail, nous présentons plus en détail la **classification** ainsi que plusieurs concepts **généraux** liés à celle-ci.

Comme nous venons de l'énoncer, le data mining permet de créer une multitude de **modèles** qui vont apporter une plus-value, de nouvelles connaissances ou encore des capacités prédictives. Dans ce travail, nous avons étudié un type particulier de modèle émanant du data mining, le **classifieur** qui est formalisé par la Définition 2.2.1.

Définition 2.2.1. *Un **classifieur** est une fonction qui assigne une classe à chaque instance d'un jeu de données*[4].

$$f : S \to C \tag{2.1}$$

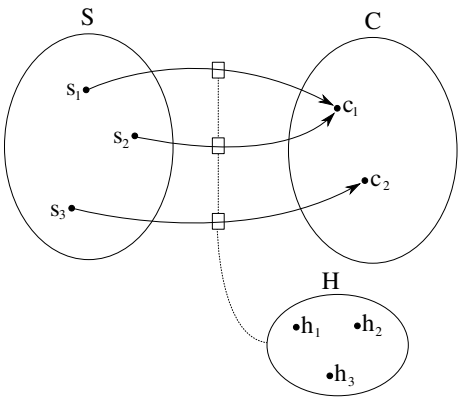

FIGURE 2.4 – Illustration du fonctionnement d'un classifieur

Dans ce document, nous nous limitons à la classification binaire, c'est-à-dire une classification où $|C| = 2$.

Afin de construire la fonction f, nous utilisons le jeu de données d'apprentissage. Cette fonction est ensuite appliquée à des instances de la "vie réelle" afin de prédire la classe à laquelle elles appartiennent.

La Figure 2.4 représente la fonction 2.1. Dans cette figure, chaque instance de l'ensemble S est liée à une classe de C grâce à f. La fonction f a été choisie

4. Cette fonction est une fonction discrète, par opposition à la régression qui construit des fonctions continues

dans l'**espace d'hypothèses** H, c'est-à-dire l'ensemble contenant toutes les fonctions réalisables. Afin de restreindre la taille de cet espace d'hypothèses, nous devons définir des contraintes sur celui-ci au moyen du **biais inductif**.

2.2.1 Biais inductif

Sans biais inductif, nous ne pouvons appliquer notre classifieur sur des jeux de données de la "vie réelle" car aucune *généralisation* n'est possible.

Il existe deux types de biais inductif [int] :

- **les biais de restriction de l'espace d'hypothèses** : limite la taille de l'espace d'hypothèses en imposant aux fonctions une structure type. Par exemple, nous choisissons de n'accepter que des formules de Forme Normale Conjonctive.
- **les biais de préférence** : présence d'un ordre de préférence dans l'espace d'hypothèses. Par exemple, le rasoir d'Occam qui préfère la f la plus simple quand il doit choisir entre deux f.

Nous pouvons également qualifier le biais choisi en le positionnant entre les deux extrêmes suivants [Mar] :

- **généralité maximum** : un ensemble d'instances est défini grâce au nombre minimum de conditions qui mènent à celui-ci.
- **spécificité maximum** : un groupe d'instances est défini grâce au nombre maximum de conditions qui mènent à celui-ci.

Illustrons cette qualification du biais au moyen d'un exemple. Imaginons deux fonctions permettant de classer des instances de jeux de données à la structure identique à celle du jeu de données de la Table 2.1. Nous hésitons entre classer toutes les instances ayant pour "Etat Civil = marié" comme positives ou classer toutes les instances ayant pour "Etat Civil = marié" *et* "Diplôme = universitaire" comme positives. Sur le jeu de données d'apprentissage, ces deux fonctions ciblent le même nombre d'instances mais lorsque celles-ci seront appliquées à la vie réelle, comme le jeu de données de la Table 2.2, la première fonction ciblera **trois** instances alors que la deuxième n'en ciblera qu'**une**. La première fonction cible potentiellement plus d'instances car celle-ci est moins restrictive et donc **plus générale**.

2.2.2 Jeux de données utilisés

Dans la section précédente, nous avons expliqué qu'un classifieur est créé à partir d'un jeu de données d'apprentissage et est ensuite appliqué à la "vie réelle", c'est-à-dire un jeu de données auquel une fonction assigne les valeurs d'attribut cible. Dans le but de pouvoir estimer la qualité du classifieur, nous devons appliquer le classifieur à un **jeu de données de test**, c'est-à-dire un jeu de données qui n'a pas servi à l'apprentissage mais qui contient les valeurs d'attribut cible. Le jeu de données de test utilisé dans le cadre de ce travail est celui de la Table 2.3. Celui-ci a une structure évidemment identique aux jeux de données des Tables 2.1 et 2.2.

Identifiant	Diplôme	Etat civil	Sexe	Enfants	Dossier approuvé ?
1	secondaire	divorcé	homme	non	non
2	universitaire	célibataire	homme	non	oui
3	primaire	divorcé	homme	oui	oui
4	secondaire	marié	homme	non	oui
5	universitaire	marié	femme	non	non
6	secondaire	divorcé	femme	non	non
7	secondaire	divorcé	femme	oui	oui
8	universitaire	marié	femme	oui	oui
9	secondaire	célibataire	homme	oui	non
10	universitaire	divorcé	homme	oui	non

TABLE 2.3 – Exemple de jeu de données de test

2.3 Arbres de décision

Après avoir décrit le data mining, nous décrivons une technique de classification appartenant à celui-ci : les **arbres de décision** [RM05]. L'arbre de décision a une structure d'arbre, telle que définie dans la théorie des graphes par la Définition 2.3.1.

Définition 2.3.1. *Un **arbre** est un graphe connexe non orienté sans cycle.*

Tout sommet d'un arbre est appelé un noeud. Un arbre est composé de noeuds particuliers, les feuilles qui sont des sommets adjacents à une seule arête. Une chaîne reliant la racine d'un arbre à une feuille est appelée une branche.

Cependant, un arbre de **décision** diffère de la Définition 2.3.1 dans le sens où celui-ci est **orienté**. Dans un arbre orienté, pour un arc donné, le noeud où l'arc est sortant est qualifié de noeud **père** tandis que le noeud où l'arc est entrant est le noeud **fils**. La **racine** de l'arbre de décision est le noeud ne possédant pas de père.

Pour classer une instance quelconque d'un jeu de données grâce à un arbre de décision, nous suivons le **chemin** allant de la racine de l'arbre vers la feuille correspondante, en respectant à chaque noeud la valeur de l'attribut sélectionné pour la segmentation.

2.3.1 Construction

La construction d'un arbre de décision à partir d'un jeu de données d'apprentissage suit toujours le même canevas [HMS66]. Nous expliquons ci-dessous en détail ce procédé de construction afin de l'enrichir et de l'adapter tout au long de ce travail.

A partir d'un ensemble d'instances S et d'un ensemble de classes C_t tel que $2 \leq t \leq q$, nous passons à l'une des étapes suivantes :

- Si S ne contient **pas d'instances**, l'arbre de décision est une feuille ayant pour classe celle définie par l'utilisateur.
- Si S ne contient que des **instances** appartenant à la **même classe** C_t, l'arbre de décision de S est une feuille ayant pour classe C_t.
- Si S contient des **instances** appartenant à **plusieurs classes**, nous **segmentons** (partitionnons) S suivant les valeurs d'un attribut. L'attribut choisi pour le partitionnement est celui qui produit la meilleure partition

suivant une mesure donnée. Le terme “meilleure partition” signifie une partition dont les sous-ensembles ont une distribution la plus **pure** possible, c'est-à-dire celle où les instances appartiennent le plus possible à une **seule classe** par sous-ensemble.
L'attribut choisi prend p valeurs $\{v_1, v_2, \ldots, v_p\}$ et partitionne S en p sous-ensembles (S_1,S_2, ..., S_p) où chaque sous-ensemble contient l'ensemble des instances ayant une valeur d'attribut particulière (exemple : S_1 est le sous-ensemble des instances ayant pour valeur d'attribut v_1). L'arbre de décision crée un noeud pour chaque S_X. Le procédé de construction est appliqué **récursivement sur l'ensemble des noeuds créés**.

Nous pouvons illustrer le processus de construction d'un arbre de décision au moyen de la Figure 2.5. Dans celle-ci, un arbre de décision composé de deux attributs et de cinq feuilles est créé. L'espace des instances, représenté par un graphique à deux dimensions [5], est également présent dans cette figure. Nous pouvons voir que durant le processus de création de l'arbre, l'espace des instances est segmenté progressivement jusqu'à atteindre cinq régions, correspondant aux cinq feuilles de l'arbre.

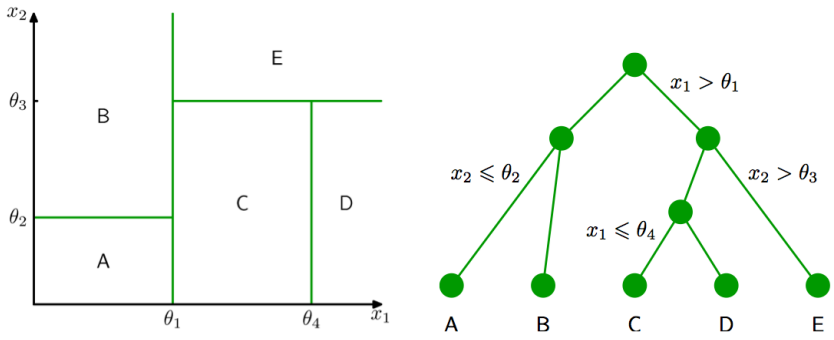

FIGURE 2.5 – Arbre et espace des instances [Car10]

2.3.2 Exemple

A partir du jeu de données présenté dans la Table 2.1, nous pouvons créer l'arbre de décision représenté dans la Figure 2.6 et qui a été appliqué au jeu de données de la Table 2.2.

5. Le jeu de données utilisé pour la construction de cet arbre est composé de deux attributs indépendants.

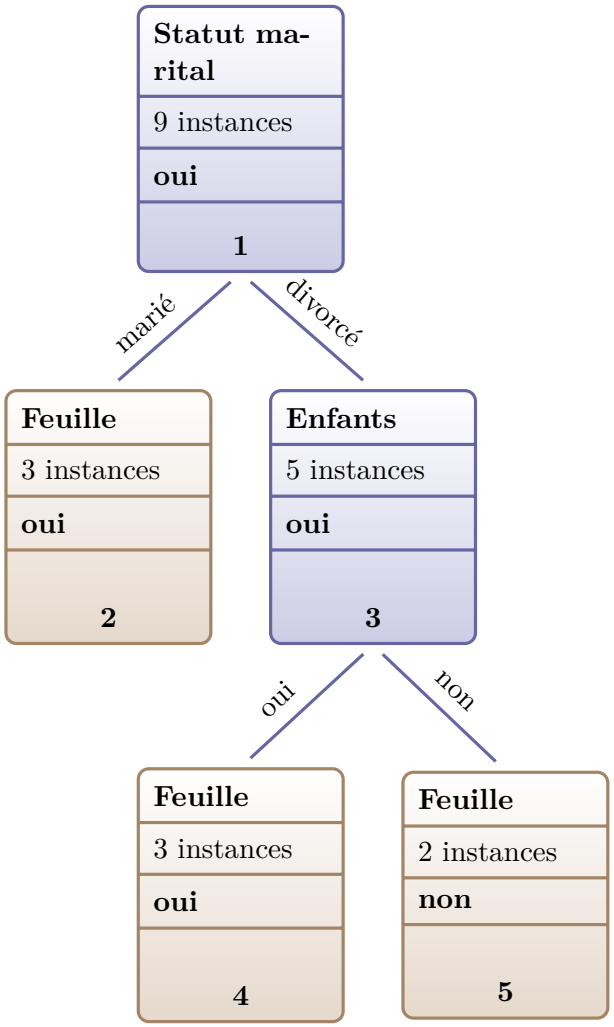

FIGURE 2.6 – Exemple d'arbre de décision construit à partir du jeu de données de la Table 2.1 et appliqué au jeu de données de la Table 2.3

Chaque **noeud** est composé de 4 informations. La **première** information indique l'attribut sélectionné pour partitionner l'ensemble d'instances pour la création du niveau suivant. La **seconde** donnée indique le nombre d'instances appartenant au noeud courant tandis que la **troisième** information contient la conclusion de chaque noeud, c'est-à-dire la classe qui sera assignée à toutes les instances appartenant à celui-ci [6]. Enfin, la **quatrième** et dernière donnée détient l'identifiant de chaque noeud de l'arbre permettant de localiser rapidement chacun de ceux-ci lors d'une description.

6. Par exemple, pour la feuille 2 de cet arbre de décision, correspondant à toutes les instances "mariées", celles-ci verront leur dossier approuvé.

Cet arbre contient deux noeuds et trois feuilles. Chaque **arc** contient la valeur de l'attribut permettant de créer le noeud/feuille suivant. Cet arbre est un arbre **partiel** car il ne permet pas de classer toutes les instances du jeu de données d'apprentissage (exemple : il ne peut classer les individus "célibataires").

2.3.3 Points clés d'un arbre de décision

Nous pouvons énoncer les trois principaux problèmes qui se posent lors de la création d'un arbre de décision :

- Comment choisir l'attribut du jeu de données qui segmentera le noeud courant ? Décrit dans la Section **2.3.4**
- Comment déterminer la bonne taille de l'arbre ? Quand arrêter le processus de construction de l'arbre ?
  Exposé dans la Section **2.3.5**
- Sur base de quels critères choisir la classe d'une feuille ?
  Décrit dans la Section **2.3.6**
- Le cas des variables continues sera traité dans la Section 2.6.

2.3.4 Segmentation

Lors de la construction d'un arbre de décision, l'action de **partitionner** un ensemble d'instances (noeud courant du niveau X) en sous-ensembles d'instances (noeuds du niveau X+1) est appelée la **segmentation**. Pour segmenter un noeud, il faut choisir un attribut. L'algorithme ayant pour tâche de construire l'arbre testera l'ensemble des attributs et choisira pour segmenter celui qui aura le meilleur résultat pour le critère choisi. Un grand nombre de méthodes de segmentation existent mais nous allons nous focaliser sur les méthodes qui sont dépendantes de la classe et qui prennent donc en compte la distribution des classes [Rou01].

Nous allons expliquer en détails deux mesures permettant de choisir l'attribut à segmenter. Les mesures expliquées sont utilisées dans la solution développée dans le cadre de ce mémoire.

Fonction d'impureté :

Une **mesure d'impureté** mesure le **désordre** dans un ensemble d'instances. Pour un noeud où toutes les instances ont la même classe (qualifié de **noeud pur**), l'impureté sera égale à zéro.
Celle-ci augmentera en fonction de l'accroissement du désordre dans un noeud,

à savoir que plus le désordre est grand, plus la valeur d'impureté est importante. Une mesure d'impureté permet également de mesurer le désordre d'une partition entière pour pouvoir choisir l'attribut qui minimisera le désordre de la partition qui est associée à celui-ci.

Pour pouvoir définir ce qu'est une mesure d'impureté, nous devons introduire certaines notations et propriétés :

- $d(C_t, T) = \frac{|\{s \in T | val_C = C_t\}|}{|T|}$ est la proportion d'instances de classe C_t dans un ensemble T, $T \subseteq S$
- $D(T) = (d(C_1, T), \ldots, d(C_q, T))$ est la distribution de l'ensemble T, composé de $|C|$ classes

Nous pouvons à présent définir la mesure d'impureté comme suit :

Définition 2.3.2. *Une mesure d'impureté est une fonction [Rou01, RM05] :*

$$f : D(T) \rightarrow R^+ \tag{2.2}$$

*qui possède les quatre **propriétés** suivantes :*

1. $f(T)$ *est minimum* $\Leftrightarrow d(C_t, T) = 1$ *pour un* $C_t \in C$
2. $f(T)$ *est maximum* $\Leftrightarrow \forall i : 1 <= t <= q, d(C_t, T) = \frac{1}{q}$
3. $\begin{cases} \textit{si pour certains ensembles d'instances } T \textit{ et } T', \\ (d(C_1, T), \ldots, d(C_q, T)) \textit{ et } (d(C_1, T'), \ldots, d(C_q, T')) \\ \textit{sont identiques alors } f(T) = f(T') \end{cases}$
4. *f est **lisse** : de faibles changements dans la distribution de la classe entraînent de faibles changements dans la mesure*

Après avoir expliqué le concept de mesure d'impureté pour un jeu de données S ainsi que ses conditions, nous pouvons appliquer cette mesure au calcul de l'impureté non plus d'un ensemble d'instances mais d'une partition $\biguplus_{i=1}^{k} T_i$ [7]. La fonction d'impureté F est la fonction d'impureté de la partition, c'est-à-dire la somme des impuretés des sous-ensembles de la partition pondérés par le pourcentage d'instances présentes dans ceux-ci.

7. La variable k indique le nombre de sous-ensembles présents dans la partition.

$$F(\biguplus_{i=1}^{k} T_i) = \sum_{i=1}^{k} \frac{|T_i|}{|T|} f(T_i) = \frac{1}{|T|} \sum_{i=1}^{k} |T_i| f(T_i) \tag{2.3}$$

Nous pouvons à présent détailler les deux mesures d'impureté qui sont utilisées dans la solution de notre travail, le **Training Set Error** et l'**entropie de Shannon**.

Training Set Error

Le Training Set Error (ou *TSE)* d'un ensemble T est une mesure qui repose sur "le nombre de **désagréments**", c'est-à-dire le pourcentage d'individus mal classés dans T. Par mal-classés, nous entendons des individus qui n'appartiennent pas à la classe majoritairement présente dans T.

L'erreur d'un ensemble d'instances T peut être calculé à partir d'une fonction E tel que

$$E(T) = 1 - \max_{C_t \in C} d(C_t, T) \tag{2.4}$$

Par exemple, dans un ensemble d'instances de distribution (7,3), la classe majoritaire (70%) de cet ensemble est la classe positive puisqu'elle est composée de sept instances alors que la classe négative n'est composée que de trois instances (30%). Nous calculons l'erreur de cet ensemble comme suit : $1 - \frac{7}{10} = \frac{3}{10}$
La Figure 2.7 montre, dans le cadre d'un attribut dépendant binaire, l'évolution de l'erreur en fonction de la distribution de l'ensemble des instances T où $d(C_t, T)$ est l'un des éléments du couple $(d(C_1, T), d(C_2, T))$.

A partir de 2.4, nous pouvons calculer l'*erreur relative*, c'est-à-dire l'erreur calculée sur une **partition**, erreur calculée comme étant l'erreur de chaque sous-ensemble de la partition pondérée par l'importance de chaque sous-ensemble au sein de la partition.

$$RE(\biguplus_{i=1}^{k} T_i) = \sum_{i=1}^{k} \frac{|T_i|}{|T|} E(T_i) \tag{2.5}$$

La version non normalisée de RE (Formule 2.5) est le TSE, calculé comme suit :

$$TSE(\biguplus_{i=1}^{k} T_i) = |T| RE(\biguplus_{i=1}^{k} T_i) \tag{2.6}$$

Entropie de Shannon

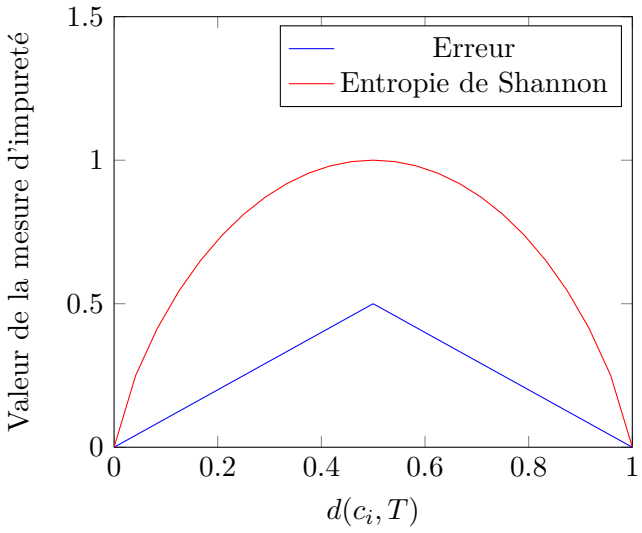

FIGURE 2.7 – Fontions d'impureté [Mar]

L'entropie de Shannon "*permet de mesurer la quantité d'information moyenne d'un ensemble d'événements (en particulier de messages) et de mesurer son incertitude*" [Az0]. Cette mesure est l'une des plus utilisées comme critère de segmentation dans les arbres de décision [Mar].

Dans ce travail, nous avons utilisé l'entropie de Shannon. Nous commençons par définir l'entropie de Shannon d'un ensemble d'instances T de distribution $((d(C_1, T), d(C_2, T), \ldots, d(C_q, T))$.

$$Ent(T) = -\sum_{t=1}^{q} d(C_t, T) \log_2(d(c_t, T)) \tag{2.7}$$

Par exemple l'entropie d'un ensemble d'instances de distribution (4,7) aura une entropie $(\frac{4}{11}) \log_2(\frac{4}{11}) - (\frac{7}{11}) \log_2(\frac{7}{11}) = 0.946$, la valeur d'entropie maximale étant atteinte pour une distribution uniforme, comme énoncé dans la propriété 2 de la Définition 2.3.2.
Comme illustré par la Figure 2.7 [8], et conformément aux propriétés des fonctions d'impureté en 2.2, l'entropie est minimale lorsque la distribution est pure et maximale lorsque la distribution indique que toutes les classes ont le même nombre d'instances.

Lorsque nous mesurons l'entropie d'une partition, nous utilisons l'**Average Class Entropy** (ACE), qui mesure la somme des entropies des sous-ensembles

8. dans le cadre d'un attribut dépendant binaire

de la partition pondérées par leur importance au sein de celle-ci :

$$ACE(\biguplus_{i=1}^{k} T_i) = \sum_{i=1}^{k} \frac{|T_i|}{|T|} Ent(T_i) \tag{2.8}$$

Après avoir choisi l'attribut permettant de segmenter le noeud (généralement l'attribut ayant la plus petite valeur d'impureté), il faut que nous déterminions en combien de feuilles partitionner le noeud courant. Encore une fois, plusieurs techniques existent, certaines adaptées à des attributs discrets, d'autres à des attributs continus :

- Certaines techniques produisent une feuille pour chaque valeur de l'attribut sélectionné pour la segmentation (exemple : C4.5).
- Certaines produisent des arbres **binaires**. Elles regroupent donc les valeurs de la variable jusqu'à ce qu'elles puissent créer deux feuilles pour le noeud courant (exemple : CART).
- Certaines vont regrouper les valeurs suivant des critères statistiques (exemple : CHAID).
- Pour les attributs de type continu, nous aborderons une technique de discrétisation largement utilisée pour ce travail dans la Section 2.6.

Lors du processus de création d'un arbre de décision, il faut prendre en compte le fait qu'un arbre de taille trop importante ne sera pas performant face à des jeux de données de test. Ce sujet est abordé dans la section suivante.

2.3.5 Problème du surajustement

Le **sous-ajustement** d'un modèle [Fee] se produit lorsqu'un modèle est trop général et possède de faibles performances à la fois par rapport aux jeux de données d'apprentissage et de test. Ce problème est néanmoins moins courant que celui du **surajustement**.
Un arbre de décision est dit **surajusté** s'il semble performant dans le sens où un très grand pourcentage d'instances du jeu de données d'apprentissage sont parfaitement classées mais que cet arbre est **inefficace en réalité** (par rapport au **jeu de données de test** et à la "vie réelle").

Toute personne inexpérimentée dans le domaine se focalisera très souvent,

lors de la création d'arbres de décision à partir de jeux de données d'apprentissage, sur le nombre d'erreurs (le nombre d'éléments mal classés) de ces arbres sur ces jeux de données.

Or, toutes les instances d'un jeu de données peuvent être correctement classées par un arbre de décision [9]. Par conséquent, n'obtenir aucune erreur de classification ne peut aucunement être considéré comme correspondant à un arbre de bonne qualité par rapport au **jeu de données de test** (ou à la "vie réelle"), mais est plutôt l'apanage d'un arbre de décision trop complexe et inapplicable à celui-ci. Un bon modèle est un arbre possédant les caractérististiques du jeu de données d'apprentissage (les informations utiles) sans avoir les spécificités de celui-ci.

Lorsque deux arbres ont les même performances sur le même jeu de données d'apprentissage, il faut sélectionner celui qui est le plus simple car il est plus stable sur le jeu de données de test [10]. Il faut donc réaliser un compromis entre la performance et la complexité du modèle.

Nous pouvons définir le surajustement de manière plus formelle à partir de [Mit97, Arb] :

Soit le **Training Set Error(M)** = le nombre d'erreurs du modèle M sur un jeu de données d'apprentissage (Pour un arbre de décision, le TSE (voir Section 2.3.4) sur la partition composée de toutes les feuilles (sous-ensembles) de cet arbre, en utilisant le jeu de données d'apprentissage)
Soit **Testing Set Error(M)** = le nombre d'erreurs du modèle M sur un jeu de données de test (Pour un arbre de décision, le TSE sur la partition composée de toutes les feuilles de cet arbre, en utilisant le jeu de données de test) et étant donné deux modèles M_1 et M_2 (par exemple deux arbres de décision) construits à partir d'un même jeu de données d'apprentissage, nous dirons que l'hypothèse M_1 est surajustée si et seulement si :

9. Le cas extrême étant que chaque feuille de l'arbre ne contienne qu'une seule instance.
10. Principe du **rasoir D'occam** [Fee] : Etant donné deux modèles ayant une erreur de généralisation (l'erreur de généralisation est l'erreur estimée sur le jeu de données de test c'est-à-dire la probabilité qu'une instance x choisie au hasard, en respectant la distribution D du jeu de données d'apprentissage soit mal-classée [MR05]) identique, on préférera le modèle le plus simple à l'autre plus complexe.

$$\begin{aligned} \text{Training Set Error}(M_1) &< \text{Training Set Error}(M_2) \\ &\wedge \\ \text{Testing Set Error}(M_1) &> \text{Testing Set Error}(M_2) \end{aligned} \tag{2.9}$$

En d'autres mots, l'hypothèse M_1 classifie les instances du jeu de données d'apprentissage de manière plus performante que l'hypothèse M_2 mais est moins performante lors de son application à de nouvelles données.

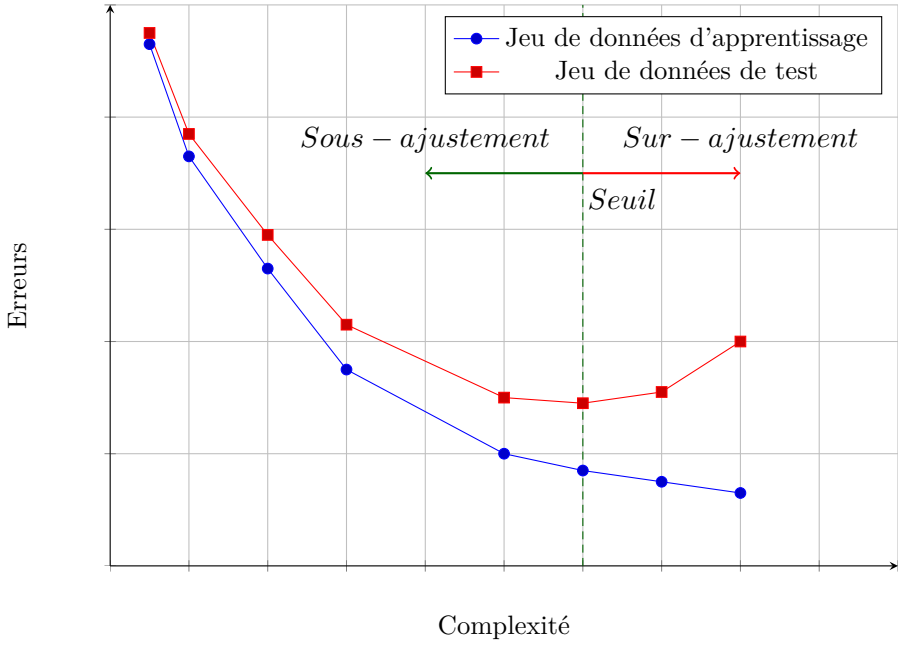

FIGURE 2.8 – Allures typiques des courbes illustrant le problème du surajustement [Mit97, Fee]

Dans le graphique de la Figure 2.8, nous observons l'évolution de l'erreur d'un arbre de décision en fonction de l'évolution de la complexité (de la taille) de celui-ci. A partir d'un certain seuil (ligne verte du graphique), alors que l'erreur du modèle sur le jeu de données d'apprentissage continue à baisser, l'erreur du modèle sur le jeu de données de test augmente, ce qui confirme le phénomène du surajustement.

Plusieurs méthodes ont été créées afin de résoudre le problème du surajus-

tement des modèles. Ces méthodes peuvent être regroupées en deux grandes catégories [Mit97] : les techniques de **pré-élagage** et les techniques de **post-élagage**.

Pré-Elagage

Contrairement à un algorithme tel qu'ID3 (Algorithme 1) qui ne s'arrête que lorsque toutes les instances du jeu de données d'apprentissage sont correctement classées, le **pré-élagage** est une technique consistant à stopper l'algorithme chargé de construire l'arbre avant qu'il n'atteigne la perfection d'un point de vue de la classification des instances du jeu de données d'apprentissage. Nous pouvons stopper l'algorithme avant que toutes les instances ne soient correctement classées selon l'une des conditions suivantes [Fee] :

- stopper l'expansion si le nombre d'instances non-correctement classées est en dessous d'un certain seuil.
- stopper l'expansion si le noeud courant n'améliore pas une mesure d'impureté d'au moins un certain seuil.
- stopper l'expansion si la pureté d'un noeud a atteint un niveau suffisant.

Cette technique présente un désavantage assez important dans le sens où l'on peut s'arrêter trop tôt dans le processus d'expansion car celle-ci ne considère que l'étape directement ultérieure avant de prendre la décision de stopper la construction du modèle [11].
De plus, le pré-élagage arrête l'expansion de l'arbre en évaluant chaque attribut de manière individuelle et néglige les interactions entre les attributs.

Pour pallier les manquements du pré-élagage, une autre technique, plus utilisée et considérée comme fournissant généralement de meilleurs résultats a été concue, il s'agit du post-élagage [Mit97].

Post-Elagage

Le **post-élagage** est une technique qui part d'un arbre complet, dont le processus d'expansion est maximal et qui élague par la suite les sous-arbres considérés comme "inutiles" [12]. Nous pouvons donner le fonctionnement général du post-élagage [Fee] :

1. Construire entièrement l'arbre. L'arbre sera par conséquent surajusté.

11. Désavantage connu sous le nom d' *horizon effect*.
12. C'est-à-dire inintéressants du point de vue des mesures utilisées pour le post-élagage.

2. Réduire les noeuds en partant des feuilles de l'arbre (processus de type "bottom-up")
3. Si l'erreur de généralisation est améliorée après l'élagage alors on remplace le sous-arbre par une feuille.
4. Assigner une classe à la nouvelle feuille.

Pour tenter d'élaguer l'arbre de façon plus performante, nous pouvons également utiliser une autre technique qui, lors de l'élagage de l'arbre, utilise un jeu de données spécialement dédié à cette tâche, le **jeu de données d'élagage**.

2.3.6 Assignation d'une classe à une feuille

Lorsque l'arbre a été construit et éventuellement élagué, nous devons affecter à chaque feuille de celui-ci une conclusion, c'est-à-dire une valeur d'attribut cible. Pour affecter une conclusion à une feuille, nous utilisons des **règles d'affectation**.
Une règle d'affectation simple est par exemple de choisir la classe majoritaire de la feuille, c'est-à-dire la classe la plus présente parmi les instances de cette feuille. Cette technique est à appliquer lorsque toutes les classes ont le même poids, la même importance. Lorsque ce n'est pas le cas, il faut se référer à d'autres techniques (Section 2.7).

2.3.7 Exemples d'algorithmes de construction d'arbres de décision

Dans cette partie, trois algorithmes de construction d'arbres de décision sont successivement présentés : **ID3**, **C4.5** et **CART**.

ID3 [Mit97]

L'Algorithme ID3 est un algorithme qui construit complètement l'arbre. Chaque noeud sélectionne un attribut qui classifie le mieux les instances du jeu de données d'apprentissage. Il répète le processus jusqu'à ce que les instances soient parfaitement classées ou jusqu'à ce que tous les attributs aient été utilisés. Ce type d'algorithme éprouve des difficultés quand il doit faire face à des jeux de données bruités ou ayant un faible nombre d'instances. Dans chaque cas, cet algorithme produira un modèle surajusté par rapport aux données du jeu de données (ID3 ne contient pas de méthode d'élagage). Le critère de segmentation de cette méthode est l'*information gain* [Rou01]. Cet algorithme permet, au même titre que C4.5 et CART une discrétisation locale [PT98]

Algorithme 1: $ID3(Examples, Target\_Attribute, Attributes)$ [Mit97]

input : *Examples* is the training dataset, *Target_Attribute* is the class value and *Attribute* is the Attribute list
output : a *decision tree* that correctly classifies the given *Examples*

create a *Root* node for the tree;
if *all Instances are positive* **then**
    return single-node tree *Root*, with label = +;
if *all Instances are negatives* **then**
    return single-node tree *Root*, with label = −;
if *Attributes is empty* **then**
    return single-node tree *Root*, with label = most common value of *Target_Attribute* in *Examples*;
begin
    $A \leftarrow$ the attribute from *Attributes* that best classifies *Examples* (with the highest information gain);
    The decision attribute for *Root* $\leftarrow A$;
    foreach *possible value* v_i *of* A **do**
        Add a new tree branch below *Root* corresponding to the test $A = v_i$;
        Let $Examples_{v_i}$ be the subset of Examples that have value v_i for A;
        if $Examples_{v_i}$ *is empty* **then**
            below this new branch add a leaf node with label = most common value of Target_Attribute in Examples;
        else
            below this new branch add the subtree
            `ID3(`$Examples_{v_i}, Target\_Attribute, Attributes - \{A\}$`);`
return *Root*;

C4.5 [Qui93]

Cet algorithme a été inventé par Quinlan en 1993. Il est basé sur ID3 mais apporte quelques améliorations non négligeables :

- gestion des attributs de valeur inconnue : l'algorithme prend en compte les instances ayant des valeurs inconnues pour certains attributs en donnant une probabilité à chaque résultat possible.
- critère de segmentation : utilisation du gain ratio [Rou01] afin de ne pas favoriser exagérément les attributs ayant un grand nombre de valeurs différentes.
- élagage de l'arbre au moyen de la méthode **Error-Based Pruning** [Qui93].
- production d'arbres de décision généralement plus compacts.

CART [Tur05]

La méthode CART (**C**lassification **A**nd **R**egression **T**rees) a été développée par Breiman [BFOS84]. Cette méthode construit des arbres de type **binaire** et utilise la mesure d'impureté "Twoing Criteria", amélioration de l'indice Gini [Rou01] pour les problèmes multi-classes ($|C| > 2$). Après avoir construit un arbre complet, une technique d'élagage nommée *Cost-Complexity* [Tur05] est employée. Celle-ci utilise un échantillon d'élagage.
Une particularité de CART est qu'elle peut générer des arbres de régression, c'est-à-dire des arbres pouvant faire face à des jeux de données présentant un **attribut cible continu**.

2.3.8 Points forts et points faibles

Les arbres de décision sont des outils intuitifs et facilement **compréhensibles** de par leur structure hiérarchique [Mar], même pour des utilisateurs n'étant pas spécialistes de ce style de classifieur [RM05]. Les arbres permettent également de décomposer des problèmes assez complexes en problèmes plus simples [Mar].
De plus, les arbres de décision peuvent gérer des attributs de types divers (continus, discrets,...). Les jeux de données utilisés pour construire les arbres peuvent avoir des valeurs manquantes ou des erreurs, certains algorithmes de construction d'arbres possèdent des techniques permettant de gérer ce genre de cas [RM05].
Les arbres de décision sont adaptés aux jeux de données de grande taille et peuvent être construits à partir de ceux-ci pour un temps d'exécution assez faible [Mar].

Ce type de classifieur possède également quelques points faibles. Premièrement, plusieurs méthodes de construction d'arbres de décision ne prennent pas en compte les attributs cibles de type continu [RM05].
Nous pouvons également citer le problème de *réplication*[13] [RM05].
De plus, un autre point faible identifié par *Quinlan* est celui de la grande sensibilité aux jeux de données d'apprentissage, aux attributs non pertinents et au bruit éventuellement présent dans les jeux de données [RM05].
Les arbres de décision sont assez sensibles au surajustement par rapport au jeu de données d'apprentissage et bon nombre d'entre eux ne sont généralement pas à l'aise en présence de jeux de données asymétriques [Mar].

13. Des sous-arbres identiques sont présents à plusieurs endroits de l'arbre.

2.4 Règles

Dans cette partie, nous allons introduire la notion de règle, détailler celle-ci et ensuite la mettre en relation avec les arbres de décision. Par la suite, nous donnerons quelques exemples de règles afin de mieux percevoir l'utilité de celles-ci.

Les **règles** permettent de représenter des sous-ensembles d'instances de manière formelle. Une règle est composée de deux parties : la partie antécédente et la partie conséquente. La règle cible un ensemble d'instances qui vérifient la **partie antécédente** de la règle (elles ont pour valeur de partie antécédente **Vrai**). Les instances qui n'appartiennent pas à l'ensemble d'instances ciblées par la règle ont la partie antécédente de la règle qui renvoie **Faux** [F.I10].
La **partie conséquente** de la règle prédit la classe des instances du jeu de données de test.

Définition 2.4.1. *Une règle i est représentée sous la forme suivante [HCGdJ10]*

$$R_i : Conditions \rightarrow Attribut_{Cible} \tag{2.10}$$

où

- ***Conditions**, la partie **antécédente** de la règle est une conjonction de conditions, c'est-à-dire une conjonction de couples de type* $(attribut, valeur)$ *[GB10]. Cette condition est de la forme suivante :*

  $$(attribut_1, valeur_1) \wedge (attribut_2, valeur_2) \wedge \ldots (attribut_{n-1}, valeur_{n-1})$$

  *est **Vraie** pour une instance quelconque i lorsque l'ensemble des valeurs des attributs* $\{attribut_1, attribut_2, \ldots attribut_{n-1}\}$ *de i sont identiques aux valeurs des attributs correspondants de la condition.*

- $\mathbf{Attribut_{Cible}}$*, la partie **conséquente** de l'implication représente la valeur de l'attribut cible attribuée à la règle [GB10].*

Exécuter l'algorithme de création du classifieur sur le jeu de données d'apprentissage permet de construire le classifieur et les règles. La partie antécédente est le chemin entre le noeud racine de l'arbre et une feuille, dans le cas d'un arbre de décision. La partie conséquente de la règle est, elle, calculée suivant

une règle d'affectation (Section 2.3.6).
Lorsque le modèle, composé d'un ensemble de règles, est appliqué à un jeu de données de test, la partie antécédente d'une règle quelconque créera le sous-ensemble d'instances tandis que la partie conséquente prédira la classe des instances du sous-ensemble.

2.4.1 Propriétés

Nous pouvons à présent introduire quelques termes [GB10] permettant de mieux caractériser les différents types de règles.
Nous dirons qu'une instance $\mathbf{x}$ d'un jeu de données est **couverte** par une règle $\mathbf{R_i}$ si et seulement si chaque paire attribut-valeur de $\mathbf{R_i}$ est satisfaite (son évaluation renvoie "vrai") par l'attribut correspondant de $\mathbf{x}$.
Une classe $\mathbf{C_t}$ sera dite **complètement couverte** par un ensemble de règles $\mathbf{R}$ si et seulement si pour chaque instance $\mathbf{x}$ de $\mathbf{C_t}$ il existe une règle $\mathbf{R_i}$ de $\mathbf{R}$ telle que cette règle couvre $\mathbf{x}$.
Un ensemble de règles $\mathbf{R}$ est dit **complet** si et seulement si chaque classe du jeu de données est complètement couverte par $\mathbf{R}$.
Une règle $\mathbf{R_i}$ est **consistante** (avec le jeu de données) si et seulement si pour chaque instance $\mathbf{x}$ couverte par $\mathbf{R_i}$, $\mathbf{x}$ est un membre de la classe $\mathbf{C_t}$ indiquée par $\mathbf{R_i}$. Un ensemble de règles $\mathbf{R}$ est **consistant** si et seulement si chaque règle de $\mathbf{R}$ est consistante avec le jeu de données.

2.4.2 Règles et arbres de décision

A partir du jeu de données de la Table 2.1 correspondant à l'arbre de décision de la Figure 2.6, nous pouvons aisément construire l'ensemble des règles du classifieur. En effet, à chaque feuille de l'arbre de décision correspond une règle. La construction d'une règle se fait donc de la manière suivante pour chaque feuille de l'arbre :

1. Sélection d'une feuille f_i, i étant l'identifiant de la feuille.
2. Pour chaque noeud situé sur le chemin entre la feuille f_i et la racine de l'arbre, **ajout de la paire** $(attribut, valeur)$ contenant l'attribut (attribut sélectionné pour la segmentation) et sa valeur associée dans l'**antécédent de la règle**.
3. Ajout du **conséquent** de la règle suivant une règle d'affectation.

Après avoir exécuté ces opérations, nous obtenons le **classifieur** suivant, présenté sous la forme de règles :

$$\begin{aligned} Statutcivil &= marié \rightarrow DossierApprouvé = oui \\ &\vee \\ Statutcivil &= divorcé \wedge Enfants = oui \rightarrow DossierApprouvé = non \\ &\vee \\ Statutcivil &= divorcé \wedge Enfants = non \rightarrow DossierApprouvé = oui \end{aligned} \tag{2.11}$$

Celui-ci est composé de trois règles qui correspondent dans l'ordre aux feuilles de l'arbre de décision.

2.4.3 Exemple

Illustrons l'utilité des règles au niveau de la visualisation à l'aide d'un exemple [NLW09]. Les règles sont appliquées à un jeu de données de test, le jeu de données de la Table 2.3. Ce jeu de données dont la structure est identique aux jeux de données des Tables 2.1 et 2.2 contient un certain nombre d'individus, chacun muni d'un identifiant. Chaque instance de ce jeu de données possède les attributs suivants :

- **indépendants** :
  - **Diplôme** : *primaire*, *secondaire* ou *universitaire*
  - **Statut** : *célibataire*, *marié* ou *divorcé*
  - **Sexe** : *homme* ou *femme*
  - **Enfants** : *oui* ou *non*
- **dépendants** :
  - **Dossier approuvé** : attribut cible, peut prendre comme valeur *oui* ou *non*

Soit le classifieur 2.11 composé de trois règles et correspondant à l'arbre de décision de la Figure 2.6. La première règle indique que chaque personne **mariée** verra son dossier **approuvé**. Le sous-ensemble associé à cette règle est donc l'ensemble des personnes mariées. La seconde règle indique que chaque personne **divorcée** et ayant des **enfants** verra son dossier **refusé**
La troisième règle indique que chaque personne **divorcée** et n'ayant **pas d'enfants** verra son dossier **approuvé**.

La Figure 2.9 représente de manière graphique les trois sous-ensembles représentés illustrant chacun une des trois règles 2.11 .

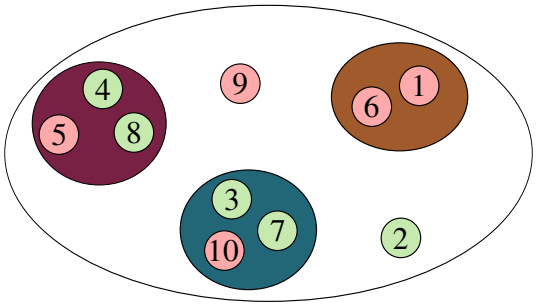

FIGURE 2.9 – Représentation des sous-ensembles d'instances correspondant aux 3 règles.

Dans le sous-ensemble de couleur rouge (règle 1), composé des instances {4, 5, 8}, nous pouvons constater que les instances {4, 8} satisfont la partie conséquente. Au contraire, l'instance {5} ne satisfait pas la partie conséquente de la règle.

Dans le sous-ensemble de couleur bleue (règle 2), composé des instances {3, 7, 10}, l'instance {10} ne satisfait pas la partie conséquente de la règle.

Dans le sous-ensemble de couleur brune (règle 3), composé des instances {1, 6}, toutes les instances satisfont la partie conséquente de la règle.

Nous pouvons à présent, à partir de cet exemple, illustrer les propriétés présentées précédemment (Section 2.4.1). Considérons chaque règle séparément :

- **Règle 1** :
  - Cette règle est associée à la classe "positive", classe composée des instances {2, 3, 4, 7, 8} et **pas** complètement **couverte** par la règle 1, celle-ci ne couvrant que les instances {4, 8}.
  - Cette règle n'est **pas consistante** car l'instance {5} n'appartient pas à la classe "positive".

- **Règle 2** :
  - Cette règle est associée à la classe "positive", classe composée des instances {2, 3, 4, 7, 8} , classe **pas** complètement **couverte** par la règle 2, celle-ci ne couvrant que les instances {3, 7}.
  - Cette règle n'est **pas consistante** car l'instance {10} n'appartient pas à la classe "positive".

- **Règle 3** :
  - Cette règle est associée à la classe “négative”, classe composée des instances {1, 5, 6, 9, 10}, classe **pas** complètement **couverte** par la règle 1, celle-ci ne couvrant que les instances {1, 6}.
  - Cette règle est **consistante** car toutes les instances de celles-ci appartiennent à la classe “négative”.

- Considérons maintenant les 3 règles **collectivement** :
  - L’ensemble des règles n’est pas **complet** car les classes du jeu de données ne sont pas complètement couvertes par cet ensemble.
  - L’ensemble des règles n’est **pas consistant** car les règles 1 et 2 ne sont **pas consistantes**.

2.5 Qualité d'un classifieur

Après avoir introduit la discipline dans laquelle s'inscrit ce travail, expliqué les arbres de décision ainsi que les règles nous présentons plusieurs formules permettant de mesurer la qualité des ensembles d'instances produits au moyen des règles. Une règle de **qualité** est une règle classifiant correctement les instances (c'est-à-dire obtenant un faible pourcentage d'instances classées de façon incorrecte), contenant un nombre d'instances suffisamment élevé, compréhensible et interprétable.

Afin de pouvoir évaluer un classifieur, nous pouvons utiliser une **matrice de confusion**[14] qui nous permettra de mesurer la qualité de la classification obtenue au moyen de celui-ci. Cette matrice est présentée par la Table 2.4. Les colonnes de cette matrice représentent la classe prédite tandis que les lignes représentent la classe actuelle des instances.

A partir d'un classifieur **binaire** et d'une instance, **quatre** résultats sont possibles. **TN** si l'instance est négative et qu'elle est classée comme négative (nommé *True Negative*), **FP** si l'instance est négative et classée par le classifieur comme positive (FP pour *False Positive*). **FN** si l'instance est positive et classée comme négative (nommé *False Negative*) et **TP** si l'instance est positive et classée par le classifieur comme positive (appelé *True Positive*) [Faw06].

Par exemple : Soit un test de dépistage d'une maladie quelconque sur un ensemble de personnes. Nous dirons qu'une personne est un **FP** si elle ne présente pas la maladie mais que le test de dépistage est positif. Une personne **FN** est une personne ayant la maladie mais un test de dépistage négatif. Un **TP** est une personne présentant la maladie et ayant un test de dépistage positif. Un patient **TN** n'a pas la maladie et son test de dépistage est négatif. Les deux cas les plus préoccupants au niveau de la qualité du classifieur correspondent aux deux cas erronés, les **FP** et les **FN**.

		Prédiction	
		Négatif	Positif
Actuel	Négatif	TN	FP
	Positif	FN	TP

TABLE 2.4 – Matrice de confusion contenant le nombre de **TN**, de **TP**, de **FN** et de **FP**

14. aussi appelée matrice/tableau de contigence

A partir des "quatres concepts" de la matrice de confusion, nous pouvons dériver toutes une série de mesures, utiles pour la qualification des règles produites par un algorithme de classification quelconque.
Un grand nombre d'autres mesures ont été conçues pour qualifier un classifieur et/ou des règles [HCGdJ10] mais nous ne présenterons que celles que nous avons effectivement utilisées dans notre solution.

Nous pouvons classer les différentes mesures ayant trait à la qualité des règles en plusieurs catégories [HCGdJ10] : complexité, généralité, précision et intérêt. Les mesures n'appartenant à aucune de ces catégories sont regroupées dans la partie "mesures hybrides".

Avant d'aborder ces différentes mesures, nous pouvons définir quelques concepts requis pour comprendre suffisamment celles-ci :

- R_i : règle numéro **i** d'un ensemble de règles.
- R : l'ensemble des règles du modèle.

Complexité

Pour qu'elles puissent être interprétées aisément, les règles doivent être simples. Deux mesures permettent de qualifier la complexité de l'ensemble des règles produit à partir d'un jeu de données et d'une technique donnés.

- le nombre de **règles** (n_r) : mesure le nombre de règles produites pour un jeu de données donné.
- le nombre d'**attributs** (n_a) : mesure le nombre d'attributs présents dans une règle. Pour un ensemble de règles, le nombre d'attributs est la moyenne du nombre d'attributs présents dans chaque règle.

Généralité

Mesure le nombre d'instances du jeu de données qui sont prises en compte par une règle particulière. Ainsi, une règle sera qualifiée de **plus générale** qu'une autre si elle comprend plus d'instances du jeu de données. Lors de la création de règles, couvrir le plus d'instances possibles pour avoir des patterns plus généraux est primordial.

- **Couverture** : Cette mesure indique le pourcentage d'instances qui satisfont la partie antécédente d'une règle [LKFT04, HCGdJ10]. Elle est

calculée comme suit :

$$Couverture(R_i) = \frac{n(Cond)}{n_s} \tag{2.12}$$

où n_s est le nombre d'instances du jeu de données et $n(Cond)$ est le nombre d'instances qui satisfont la partie antécédente de R_i.

La **couverture moyenne** d'un ensemble de règles est calculée comme suit [LKFT04] :

$$COUVERTURE(R) = \frac{1}{n_r} * \sum_{i=1}^{n_r} Cov(R_i) \tag{2.13}$$

Précision

Ce type de mesure calcule la précision de la classification obtenue à partir d' un ensemble de règles données. Par précision, nous entendons le nombre d'instances correctement classées.

- **Accuracy** : Mesure le nombre d'instances correctement classés par le classifieur (le nombre de "true"). Cette mesure est calculée comme suit :

$$Accuracy(R) = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.14}$$

- **Confiance** : Mesure le nombre d'instances vérifiant la règle complète (*antécédent* et *conséquent*) par rapport au nombre d'instances satisfaisant simplement l'*antécédent*. Cette mesure est calculée comme suit :

$$Confiance(R_i) = \frac{n(Cible.Cond)}{n(Cond)} = \frac{TP}{n(Cond)} \tag{2.15}$$

où $n(Cible.Cond) = TP$ est le nombre d'instances qui satisfont l'antécédent et ont comme valeur d'attribut cible "positif". Cette mesure est également appelée *precision* ou encore **rule accuracy**.

Hybrides

- **Sensibilité** : Mesure le pourcentage de **TP** parmi le nombre d'instances ayant pour valeur de variable cible "Positif". Elle combine la précision et

la généralité.

$$Sensibilité(R_i) = TPR = \frac{n(Cible.Cond)}{n(Cible)} = \frac{TP}{P} = \frac{TP}{TP + FN} \quad (2.16)$$

où TPR signifie *True Positive Rate* et P est le nombre d'instances compris dans une règle ayant pour valeur de variable cible "Positif". Aussi appelé *recall*.

– **False Alarm** : Mesure la proportion d'instances négatives classées de façon erronée.

$$FalseAlarm(R_i) = FPR = \frac{n(\overline{Cible.Cond})}{n(\overline{Cible})} = \frac{FP}{N} = \frac{FP}{FP + TN} \quad (2.17)$$

où $N = n(\overline{Cible})$ est l'ensemble des instances qui ont pour valeur de variable cible *négatif*, FPR signifie *False Positive Rate* et où $n(\overline{Cible.Cond})$ correspond aux instances ayant "Négatif" pour valeur de variable cible et ne vérifiant par "Cond".

– **Spécificité** : Mesure le pourcentage de **TN** parmi le nombre d'instances ayant pour valeur de variable cible "Négatif"

$$spécificité(R_i) = \frac{n(\overline{Cible.Cond})}{n(\overline{Cible})} = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR \quad (2.18)$$

– **Unusualness (WRAcc)** Mesure qui fournit un compromis entre la généralité et l'accuracy relative. Elle est exprimée de la façon suivante :

$$WRAcc(R_i) = \frac{n(Cond)}{n_s} . \left(\frac{n(Cible.Cond)}{n(Cond)} - \frac{n(Cible)}{n_s} \right) \quad (2.19)$$

La WRACC du classifieur peut-être obtenue en faisant une moyenne des WRACC de chaque règle.

2.5.1 Espace ROC

L'espace ROC est une mesure de plus en plus utilisée ces dernières années. De multiples études montrent que l'accuracy est une métrique de faible qualité pour mesurer les performances des ensembles d'instances produits par les règles. [Faw06].

L'espace ROC est la combinaison de deux mesures qui permet une inter-

prétation graphique simple des résultats. Elle est utilisée pour visualiser et sélectionner un classifieur sur base des performances de ce dernier. Ce type d'interprétation est utilisé dans de nombreux domaines [Faw06].

Cette mesure permettant une interprétation graphique donne la performance du classifieur sur un (ensemble de) compromis entre les **TP** (les *bénéfices*) et les **FP** (les *coûts*). Un exemple d'espace ROC est illustré par la Figure 2.10. L'axe **X** d'un espace ROC représente le pourcentage de **FP** (Formule 2.16) et l'axe **Y** de l'espace ROC représente le pourcentage de **TP** (Formule 2.17).
L'espace ROC permet donc de voir les performances du classifieur sur le compromis entre le nombre d'instances de classe positive classées correctement parmi toutes les instances positives du jeu de données *et* le nombre d'instances de classe négative classées incorrectement parmi l'ensemble des instances de classe négative.

Interprétation

Pour interpréter un espace ROC, nous pouvons expliquer plus en profondeur quelques éléments clés de celui-ci [Faw06] :

- le point $(\mathbf{0}, \mathbf{0})$: Ce point correspond à un classifieur qui ne fait jamais de classification positive (toutes les instances sont classées comme **négatives**), il ne commet pas d'erreur de **FP**, mais il n'a pas non plus de **TP**.

- le point $(\mathbf{0}, \mathbf{1})$: C'est le point qu'il faut essayer d'atteindre. Un classifieur sur ce point correspond à 100% de sensibilité (pas de **FN**) et pas de **FP**. Ce point correspond à la classification parfaite.

- le point $(\mathbf{1}, \mathbf{0})$: toutes les instances sont classées de façon incorrecte.

- le point $(\mathbf{1}, \mathbf{1})$: Celui-ci représente un classifieur qui classe toutes les instances de façon positive. Le classifieur a 100% de **TP** mais également 100% de **FP**.

- ligne de **non-discrimination** : Elle joint le coin inférieur gauche au coin supérieur droit. Elle a pour but de diviser l'espace ROC en deux parties. La partie située au-dessus de la diagonale correspond aux "bons" résultats de classification tandis que la partie inférieure comprend les "mauvais" résultats de classification.
  Tous les classifieurs situés sur cette ligne sont équivalents à la stratégie

choisissant au hasard la classe d'une instance. Par exemple, si un classifieur devine la classe positive la moitié du temps, il sera positionné sur le point $(0.5, 0.5)$ de l'espace ROC. Un classifieur situé sur cette ligne produira donc un point ROC qui bouge sur la ligne de non-discrimination suivant la fréquence avec laquelle il devine la classe positive.

Intuitivement, nous pouvons dire qu'un point **A** de l'espace ROC est "meilleur" qu'un point **B** si celui-ci est situé plus au nord-ouest que le point **B** car une position plus au nord-ouest a un **TPR** plus haut et un **FPR** moins élevé.
Les points situés près de l'axe x peuvent être qualifiés de "conservateurs". Ils classent des instances de façon positive uniquement quand ils sont convaincus que c'est la bonne classification et font par conséquent peu d'erreur de type **FP**, mais ont souvent un **TPR** assez bas.
A contrario, les points situés "en haut" de l'espace ROC peuvent être qualifiés de "libéraux" car ils classent des instances de façon positive alors qu'ils n'en sont pas convaincus et font par conséquent quelques erreurs de type **FP** mais ont souvent un **TPR** assez haut.

Chaque classifieur qui est situé en dessous de la ligne de non-discrimination a une performance qui est plus basse que le *hasard*. Cette zone est par conséquent généralement vide. Néanmoins, tous les points situés dans cette zone peuvent être inversés. Si nous renversons les décisions de classification pour chaque instance (toutes les classifications positives deviennent négatives et inversement), un classifieur situé dans la zone du bas sera, après inversion, situé dans la zone du haut et aura de meilleures performances de classification.

Exemple d'espace ROC

Pour illustrer l'utilité d'un espace ROC, il faut se baser sur un ensemble de matrices de confusion (voir Table 2.4). Nous allons créer un espace ROC contenant les quatre classifieurs suivants :

	Négatif	Positif
Négatif	TN = 72	FP = 28
Positif	FN = 37	TP = 63

Classifieur A

	Négatif	Positif
Négatif	TN = 23	FP = 77
Positif	FN = 23	TP = 77

Classifieur B

Table 2.5 – Matrices de confusion des classifieurs A et B

Ces quatre matrices de confusion sont respectivement représentées par quatre

	Négatif	Positif
Négatif	TN = 12	FP = 88
Positif	FN = 76	TP = 24

Classifieur C

	Négatif	Positif
Négatif	TN = 88	FP = 12
Positif	FN = 24	TP = 76

Classifieur C'

TABLE 2.6 – Matrices de confusion des classifieurs C et C'

points, **A**, **B**, **C** et **C'** dans l'espace ROC de la Figure 2.10.

Le classifieur **A** fournit la meilleure classification parmi **A**, **B** et **C**. Le point représentant le classifieur **B** est situé sur la ligne de non-discrimination et a une performance qui correspond au hasard.
Le point **C** a une performance inférieure au hasard et si l'on inverse C, nous obtenons **C'** . Ce classifieur a des performances meilleures que **A**, **B** et **C**. Inversé veut dire que lorsque **C** prédit positif, **C'** prédit négatif et inversement.

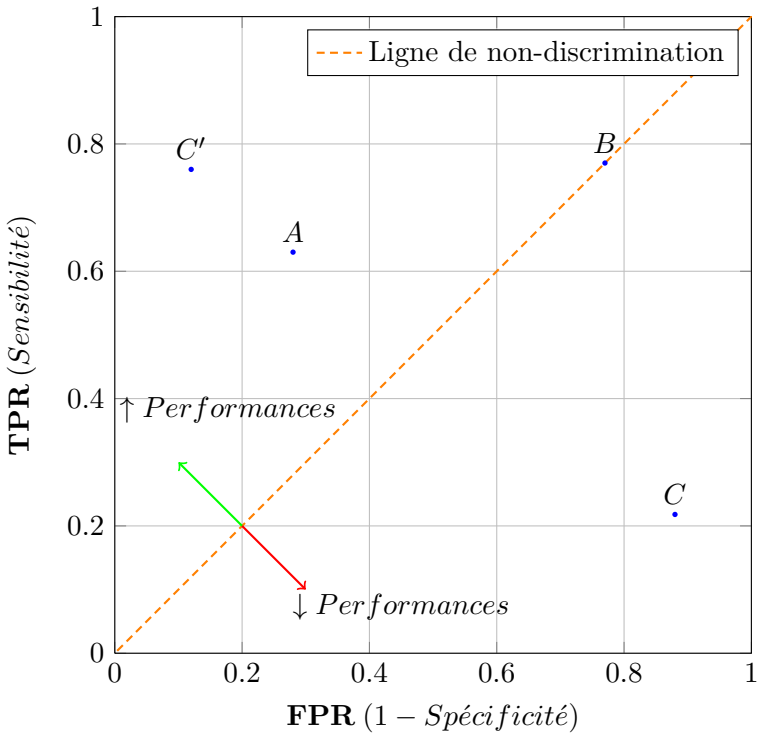

FIGURE 2.10 – Espace ROC contenant les quatre classifieurs

Courbe ROC

Nous avons jusqu'à présent utilisé des classifieurs qui assignent à une instance une des valeurs de l'attribut cible, appelés des **classifieurs discrets**. Lorsqu'un classifieur discret est appliqué à un jeu de données de test, une ma-

trice de confusion unique est produite, ce qui correspond à un seul point de l'espace ROC.
D'autres types de classifieurs assignent à chaque instance, non pas une une classe, mais une probabilité d'appartenance à la classe. Ce type de classifieur est appelé **classifieur probabiliste**.
Un classifieur probabiliste peut construire une **courbe ROC**, c'est-à-dire un ensemble de points reliés (un point correspondant à une matrice de confusion) en faisant varier le seuil de décision, c'est-à-dire en faisant varier la probabilité minimale d'appartenance à la classe positive.

Imaginons un jeu de données tel que celui représenté par la Table 2.7. Ce jeu de données de test contient 20 instances, chaque instance possédant sa classe réelle ainsi que la probabilité d'appartenance à la classe positive (représentée par un p [15]) calculée par un classifieur probabiliste. A partir de ce jeu de données, nous construisons la courbe ROC de la Figure 2.11. Chaque point de cette courbe est la matrice de confusion du jeu de données pour un seuil de décision particulier [16]. Par exemple, la probabilité 0.505 signifie que, selon le classifieur utilisé, l'instance 10 à une probabilité d'appartenance à la classe positive de 0.505. Lors du choix de ce seuil de décision, nous construisons une matrice de confusion où nous assignons la classe positive à toutes les instances ayant **au moins** cette probabilité d'appartenance et nous assignons la classe négative aux instances restantes. Le point correspond aux valeurs de **TPR** et **FPR** calculés à partir de la matrice de confusion correspondante.

Pour chaque seuil, nous construisons une matrice de confusion et calculons ensuite le **TPR** et **FPR** de celle-ci afin d'obtenir des points sur l'espace ROC. Nous relions ensuite l'ensemble des points afin d'obtenir une courbe ROC [VC06].

Les méthodes de transformation d'un classifieur probabiliste en classifieur binaire et inversement sont détaillées dans [Faw06].

Aire sous la courbe (AUC)

Afin de comparer plusieurs classifieurs de manière précise, nous calculons l'**aire sous la courbe**. Nous laissons au lecteur le soin de parcourir des articles tels que [Faw06, VC06] afin de s'informer des mesures permettant de calculer cette aire.

15. la classe négative est représentée par un n
16. Un point est créé pour chaque valeur distincte de probabilité d'appartenance (de seuil de décision).

Instance	Classe	Pr.
1	p	0.9
2	p	0.8
3	n	0.7
4	p	0.6
5	p	0.55
6	p	0.54
7	n	0.53
8	n	0.52
9	p	0.51
10	n	0.505

Instance	Classe	Pr.
11	p	0.4
12	n	0.39
13	p	0.38
14	p	0.37
15	n	0.36
16	n	0.35
17	p	0.34
18	n	0.33
19	p	0.30
20	n	0.1

TABLE 2.7 – Jeu de données de test après application d'un classifieur probabiliste [17] [VC06]

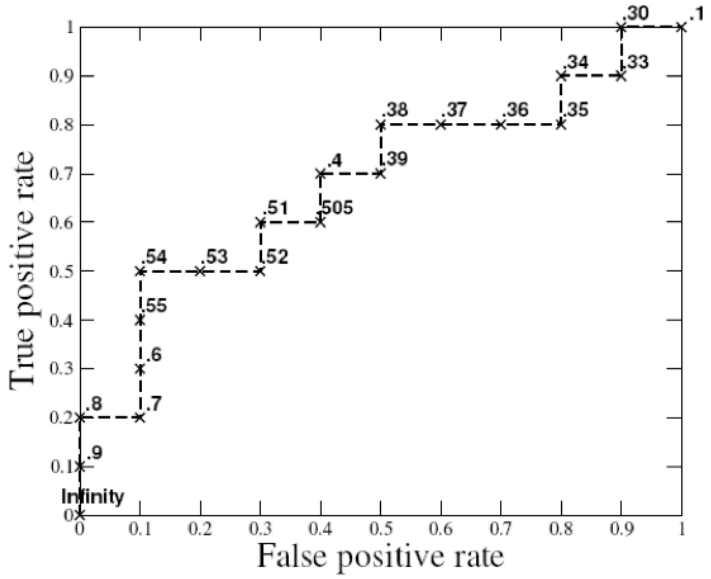

FIGURE 2.11 – Exemple de courbe ROC construite à partir du jeu de données de la Table 2.7

2.6 Discrétisation

Lors de la création d'un arbre de décision, un noeud père est partitionné en un ensemble fini de noeuds fils suivant les valeurs de l'attribut sélectionné pour la segmentation. Lorsque le nombre de valeurs de l'attribut est restreint, la segmentation est relativement aisée et compréhensible. Il suffit généralement de créer un noeud fils pour chaque valeur de l'attribut, ou encore un noeud fils pour un sous-ensemble des valeurs de l'attribut.

Lorsque le jeu de données contient des données **quantitatives**, le processus de segmentation est dès lors plus **complexe**. En effet, de nouveaux problèmes, de nouvelles questions voient le jour telles que le nombre de noeuds fils créés.
Le procédé permettant de partitionner un attribut quantitatif en un nombre fini de sous-ensembles est la **discrétisation**.
Cette partie précise les notions préalables à la bonne compréhension de la discrétisation avant de définir plus précisément celle-ci ainsi qu'un processus de base.

2.6.1 Discrétisation : intuition

Avant d'expliquer formellement la discrétisation, nous introduisons celle-ci au moyen d'un exemple.
Imaginons un jeu de données composé d'individus caractérisés par leur nom, leur prénom ainsi que leur âge. L'attribut cible de ce jeu de données est un attribut cible au sens similaire à celui du jeu de données de la Table 2.1, c'est-à-dire l'approbation d'un dossier selon les caractéristiques de l'individu. Les valeurs de l'attribut âge de tous les individus de ce jeu de données sont exposées dans la Figure 2.12 (identifiés par une lettre majuscule et colorés suivant l'approbation de leur dossier (vert) ou non (rouge)). Dans celle-ci, nous pouvons découvrir que le jeu de données contient neuf individus ayant un âge compris entre neuf et 20 ans.

A partir de cette représentation de l'attribut quantitatif âge, nous voulons transformer celui-ci afin de pouvoir regrouper les individus du jeu de données suivant cet attribut en des sous-ensembles "cohérents" par rapport à l'attribut cible, c'est-à-dire des sous-ensembles de distribution la plus pure possible [18], tout en limitant le nombre de sous-ensembles produits. Dans ce but nous créons trois groupes d'individus : les *pré-adolescents*, les *adolescents* ainsi que les *adultes*. Ces groupes ainsi que leurs intervalles sont illustrés par la Figure 2.12.

Détaillons à présent les caractéristiques des groupes que nous venons de créer. Ceux-ci correspondent aux intervalles suivants :

- $\leftarrow; 13]$: cet intervalle contient tous les individus ayant moins de 13 ans, c'est-à-dire les individus {D, E, K}.
- $]13; 18]$: cet intervalle contient tous les individus ayant entre 13 (non inclus) et 18 ans, c'est-à-dire les individus {A, B, F, H }.

18. Dans cet exemple, distribution pure signifie un sous-ensemble ne contenant que des individus dont le dossier est approuvé ou uniquement des individus dont le dossier n'est pas approuvé.

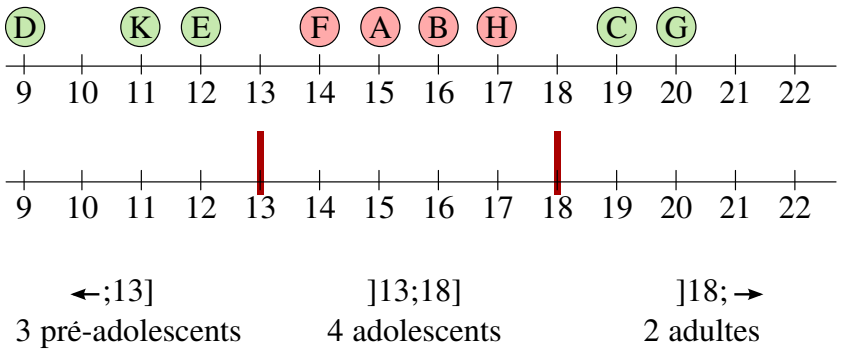

FIGURE 2.12 – Groupement des individus suivant leur âge

– $]18;\rightarrow$: cet intervalle contient tous les individus ayant plus de 18 ans, c'est-à-dire les individus {C, G}.

En regroupant les individus suivant leur âge, ceux-ci appartiennent désormais à l'un des trois sous-ensembles d'individus suivants : "pré-adolescents", "adolescents", "adulte". Ce groupement a été construit, comme dit précédemment dans le but que la distribution de chaque sous-ensemble produit soit la plus pure possible. Le processus de création de sous-ensembles d'instances suivant une valeur d'attribut se nomme la discrétisation. Dans le cas où celle-ci prend en compte la classe de chaque instance, comme c'est le cas dans cet exemple, celle-ci est dite **supervisée**.

2.6.2 Discrétisation : formalisation

En discrétisant, nous produisons, à partir d'un attribut quantitatif, une partition composée de k sous-ensembles [19]. Le nombre de sous-ensembles de la partition est, pour la discrétisation, dénommé arité. L'**arité** de la discrétisation est directement liée à un autre concept, celui de cut point. Chaque sous-ensemble de la partition produite correspondant à un intervalle de valeurs et les cut points ($k-1$ cut points) sont les nombres bornant ces intervalles. Un **cut point** est initialement présent entre chaque valeur de l'attribut à discrétiser, comme illustré par la Figure 2.13 contenant les cut points $\{u_1, u_2, \ldots, u_8\}$. C'est en déterminant l'arité de discrétisation que nous choisirons les $k-1$ meilleurs cut points, les cut points u_1 et u_7 pour l'exemple de la Figure 2.13.

Après avoir introduit la notion de discrétisation, nous nous focalisons à présent sur le choix des "meilleurs" cut points. Ce choix est effectué au moyen d'une

19. La partition de l'exemple précédent contient trois sous-ensembles.

fonction d'impureté, la partition choisie étant celle qui minimise l'impureté.

La discrétisation se formule de la manière suivante [Rou01] :

Soit
- une fonction d'impureté F,
- un ensemble d'instances T,
- un attribut quantitatif A,
- un ensemble de cut points $U = \{u_1, u_2, \ldots, u_{b-1}\} \subseteq Dom(A), u_1 < u_2 < \ldots < u_{b-1}$ qui partitionnent T en b sous-ensembles indivisibles $T_1, \ldots, T_b$,
- un entier $k \in \{1, \ldots, b\}$
- $P(U,T)$ une partition créée à partir des cut points U,

Nous pouvons donner deux définitions de la discrétisation suivant le **paramètre d'arité** utilisé :

Définition 2.6.1. *Une **discrétisation d'arité bornée** k aura une arité **maximale** de k. Discrétiser suivant une arité bornée consiste donc en la recherche d'un ensemble de cut points $U^* \subset U$, de taille maximum $k-1$ tel que : $F(P(U^*,T)) \leq F(P(U',T))$ pour tout $U' \subset U, |U'| < k$.*

Définition 2.6.2. *Une **discrétisation d'aritée fixe** k aura une arité d'**exactement** k. Discrétiser suivant une arité fixe consiste donc en la recherche d'un ensemble de cut points $U^* \subset U$, de taille $k-1$ tel que :*
$F(P(U^*,T)) \leq F(P(U',T))$ pour tout $U' \subset U, |U'| = k-1$.

Lors de l'application d'une méthode de discrétisation à un jeu de données, l'évaluation de chaque cut point prend un temps considérable. Afin de réduire celui-ci, plusieurs études ont montré qu'il était suffisant de se concentrer uniquement sur un sous-ensemble de cut points [FI93, Rou01] : les **boundary points**.

Définition 2.6.3. *[BV98] Une valeur b appartenant au domaine de l'attribut A est un **boundary point** si et seulement si dans l'ensemble des instances classées suivant leur valeur d'attribut A, il existe deux instances s_1 et $s_2 \in T$ qui ont des classes différentes telles que $val_A(s_1) < b < val_A(s_2)$ et il n'existe aucune autre instance s' tel que $val_A(s_1) < val_A(s') < val_A(s_2)$*

Pour la Figure 2.13, deux boundary points sont présents entre des instances de valeur d'attribut cible différente, les boundary points b_1 et b_2.

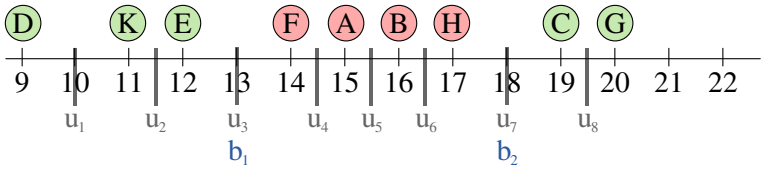

FIGURE 2.13 – Exemples de cut points et de boundary points

2.6.3 Taxonomie

Les différentes méthodes de discrétisation étant nombreuses et variées, nous pouvons tenter de classer celles-ci selon différents critères [YWW10] [20] :

- **Supervisée** et **non supervisée** : Les méthodes qui **utilisent la classe** des instances pour choisir les différents cut points nécessaire à la discrétisation sont dites supervisées. Au contraire, les méthodes qui ne tiennent pas compte de la classe sont dites non supervisées.

- **Paramétrique** et **non paramétrique** : Une discrétisation paramétrique **requiert des données**, introduites par l'utilisateur (comme l'*arité*). Une discrétisation non paramétrique n'a pas besoin de données de l'utilisateur et se base uniquement sur le jeu de données.

- **Hiérarchique** et **non hiérarchique** : Une discrétisation hiérarchique sélectionne les cut points de façon incrémentale. La discrétisation hiérarchique peut être de deux types : **top-down** ou **bottom-up**.
  La discrétisation hiérarchique top-down démarre avec un intervalle unique possédant toutes les valeurs et **segmente** ensuite cet intervalle en sous-intervalles jusqu'à ce que le critère d'arrêt soit atteint. La discrétisation hiérarchique bottom-up contient au départ autant d'intervalles que de valeurs de l'attribut. Ensuite, on **fusionne** les intervalles jusqu'à ce que le critère d'arrêt soit rencontré.
  La discrétisation non hiérarchique ne forme pas de hiérarchie durant le processus de discrétisation. Un exemple de discrétisation non hiérarchique est celui d'une méthode qui parcourt les valeurs une seule fois pour créer les intervalles.

- **Univariée** et **multivariée** : Les méthodes qui discrétisent chaque attri-

20. Pour une classification plus détaillée des différentes méthodes de classification : Annexe A.2.

but indépendamment sont univariées, tandis que celles qui prennent en compte les relations entre les attributs pour la discrétisation sont multivariées.

– **Disjointe** et **non disjointe** : Les méthodes de discrétisation disjointes discrétisent en intervalles disjoints, contrairement aux méthodes de discrétisation non disjointes.

– **Globale** et **Locale** : Les méthodes de discrétisation globales discrétisent sur base du jeu de données initial, tandis que les méthodes de discrétisation locales utilisent un sous-ensemble du jeu de données pour discrétiser "*Par exemple, nous pourrons discrétiser plusieurs fois un même attribut à différents endroits de l'arbre de décision* "[Qui93].

2.6.4 Processus de base

Nous allons à présent exposer un processus de discrétisation de base [21] afin de donner une meilleure compréhension de celui-ci.

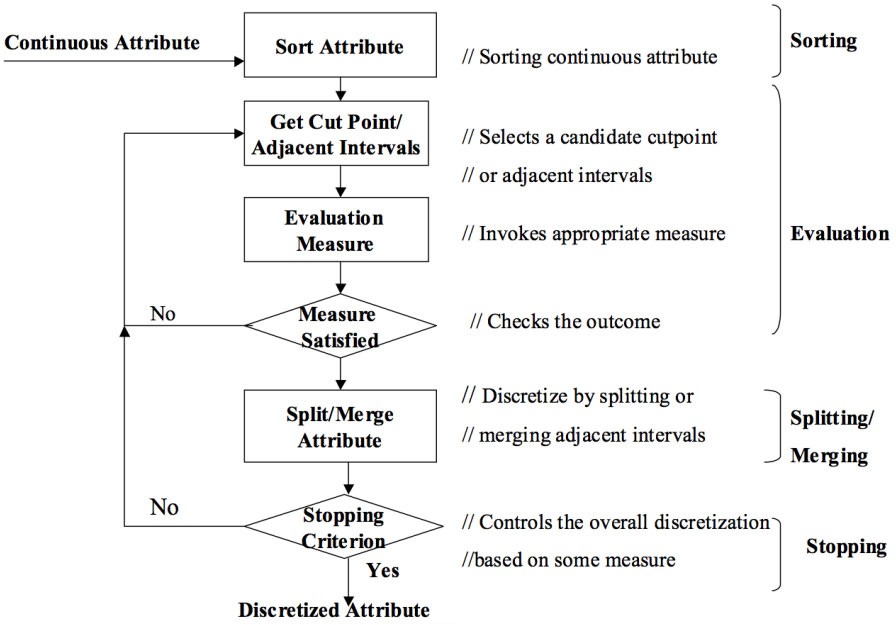

FIGURE 2.14 – Exemple de processus de discrétisation

21. Ce processus donne une idée générale du fonctionnement d'une méthode de discrétisation et n'a pas pour objectif de couvrir le fonctionnement de l'ensemble des méthodes existantes

Ce processus contient quatre étapes :

1. La première étape du processus de discrétisation est le **tri** des **instances** selon l'attribut à discrétiser afin de pouvoir évaluer celles-ci de manière chronologique. Ensuite, nous construisons l'ensemble des cut points (boundary points si la fonction d'impureté est compatible et si discrétisation supervisée) potentiels.
2. La deuxième étape teste chaque cut point afin de déterminer lequel a la meilleure valeur de fonction d'impureté.
3. Après cette étape, nous devons **partitionner** l'ensemble (si on est face à une discrétisation de type top-down) suivant le cut point choisi ou **fusionner** deux intervalles adjacents suivant le cut point choisi pour la fusion (dans le cas d'une discrétisation de type bottom-up, voir ci-dessous)
4. Nous répétons ce processus[22] (en repartant de l'étape 2 - évaluation) jusqu'à ce que le **critère d'arrêt** soit atteint. "*Celui-ci va déterminer quand stopper le processus de discrétisation. Il fait généralement un compromis entre une arité de petite taille, aisée à comprendre mais ayant une faible accuracy et une arité de grande taille, plus difficile à comprendre mais possédant une bonne accuracy. Un exemple de critère d'arrêt est le principe de description de longueur minimale*" [LHTD02]. Un autre critère d'arrêt est l'arité bornée k qui va choisir la meilleure discrétisation, d'arité maximale k [LHTD02].

2.6.5 Utilités de la discrétisation

- **points intéressants** [min] : Le processus de discrétisation permet de cibler des valeurs (cut points) intéressantes : des valeurs qui divisent potentiellement l'attribut en des sous-ensembles d'instances au propriétés différentes.
- **compatibilité** [DSC] : De nombreuses méthodes du data mining ne savent pas appréhender les attributs quantitatifs. Les attributs quantitatifs doivent donc être discrétisés en attributs qualitatifs.
- **attribut cible quantitatif** [DSC] : Lorsque l'attribut cible d'un jeu de données est un attribut quantitatif, celle-ci peut être discrétisé afin d'être divisé en plusieurs régions.
- **simplicité** [DSC] : La discrétisation simplifie et réduit la taille des données. Celles-ci peuvent par conséquent être plus facilement comparées, examinées et utilisées.

22. Ce processus n'est donc pas un processus hiérarchique.

2.7 Asymétrie

Lorsque nous sommes confrontés à des jeux de données dans le cadre d'un apprentissage supervisé, nous n'accordons généralement pas la même importance à chaque classe du jeu de données. Lorsque les jeux de données sont asymétriques tel qu'un jeu de données contenant des transactions électroniques [CS98], la classe la plus importante à détecter est celle des transactions frauduleuses. Celles-ci sont également les plus difficiles à détecter, vu le faible pourcentage d'instances de cette classe. De même, lorsqu'il s'agit de diagnostiquer des malades de la thyroïde [datc], de rechercher des gisements de pétrole sur images satellites [KHM98] ou encore de prédire la défaillance d'un matériel de télécommunication [WH98], c'est la classe minoritaire du jeu de données qui doit accaparer la plus grande part d'attention.

Ce chapitre expose les difficultés liées à l'apprentissage à partir de jeux de données asymétriques ainsi que quelques techniques permettant d'améliorer leur prise en compte.

Nous commençons par définir ce que nous entendons précisément par "rareté". En data mining, le terme rare peut prendre deux significations différentes. D'une part le terme **classe rare** et d'autre part le terme **cas rare**.

Définition 2.7.1. *[Wei04] Une classe d'un jeu de données est appelée une* ***classe rare*** *s'il n'y a qu'un faible nombre d'instances qui appartiennent à celle-ci. Une classe rare d'un jeu de données binaire est appelée classe minoritaire. La distribution d'un jeu de données possédant une classe rare est donc très inégale, une classe étant largement sous-représentée.*

Un exemple de classe rare dans un jeu de données est celui permettant de rechercher des gisements de pétrole sur images satellites [KHM98]. Dans celui-ci, 41 images sur 937 contiennent des gisements de pétrole. Nous pouvons dire que la classe comprenant les images contenant un gisement est une classe rare, la distribution de forme (contenant, contenant pas) étant de (4.19%, 95.81%).

La deuxième signification que peut prendre le terme de rareté est celui de cas rare.

Définition 2.7.2. *[Wei04] Un* ***cas rare*** *couvre une petite région de l'espace des instances et contient un très faible nombre d'instances. Un cas rare dépend*

seulement de la distribution des données et peut être présent dans des jeux de données contenant un attribut cible ou non. Dans le cas de données supervisées, un cas rare correspond à une sous-classe, un sous-concept qui apparaît rarement.

Nous pouvons citer comme exemple une technique de classification travaillant sur des jeux de données de patients dont certains sont atteints d'un cancer. Pour cet exemple, un cas rare peut être celui d'une forme rare de cancer.

Dans ce travail, nous nous focalisons sur les classes rares (sur les jeux de données asymétriques) bien plus que sur les cas rares mais nous présentons néanmoins les deux concepts car ceux-ci sont très liés, comme le montrent plusieurs études [Wei04]. En effet, nous pouvons par exemple dire que les classes minoritaires ont généralement une plus grande proportion de cas rares que les autres. D'après [Wei04], les deux formes de rareté peuvent être résolues en utilisant des méthodes similaires.

Nous pouvons nous faire une idée plus précise des concepts de classe rare et de cas rare au moyen de la Figure 2.15. Celle-ci représente l'espace des instances d'un jeu de données composé de deux classes, une classe positive ("+") et une classe négative ("-"). La distribution du jeu de données de forme (classe positive, classe négative) est (17, 44). Nous sommes par conséquent dans le cas d'un jeu de données asymétrique, ayant pour **classe rare** la **classe positive**. Nous pouvons à présent subdiviser l'ensemble des instances positives en trois régions : P1, P2 et P3. P1 regroupe une grande partie des instances positives et n'est donc pas problématique lors de l'apprentissage par un classifieur. A contrario, P2 et P3 contiennent seulement quelques instances et sont donc considérées comme des **cas rares**.

Nous pouvons à présent illustrer les conséquences qu'ont les classes/cas rares d'un jeu de données sur l'apprentissage. La Figure 2.16 représente l'espace des instances d'un jeu de données qui contient deux classes et qui a une distribution de forme (classe positive, classe négative). La classe positive est la classe minoritaire de ce jeu de données.
Les traits pleins de cette Figure représentent les bornes correctes de décision (bornes idéales) d'une classification réelle alors que les bornes d'apprentissage sont représentées par des traits pointillés. Les cinq sous-ensembles contenant des instances positives sont nommés {A1, A2, A3, A4, A5}. Les deux sous-ensembles contenant des instances négatives sont nommés {B1, B2}. Les ensembles {A1, A2, A3, A4, A5} appartiennent donc à la classe minoritaire alors que les ensembles {B1, B2} appartiennent à la classe majoritaire. $A1$ est un cas commun

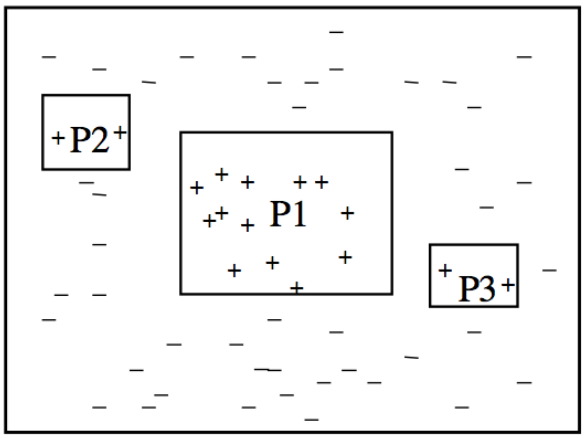

FIGURE 2.15 – Rareté dans un jeu de données [Wei05]

(qui comprend un nombre assez important d'instances), au contraire des sous-ensembles $A2, A3, A4, A5$. Ceux-ci sont des cas rares de la classe minoritaire. $B1$ est un cas commun et comprend donc un nombre important d'instances, contrairement à $B2$ qui est un cas rare. Grâce aux bornes de cette Figure, nous visualisons que les bornes d'apprentissage sont moins précises (moins en accord avec les bornes idéales) lorsqu'elles font face à des cas rares. A ce titre, les sous-ensembles $A3, A4, A5$ peuvent être qualifiés de *Small Disjuncts* [Gom07].

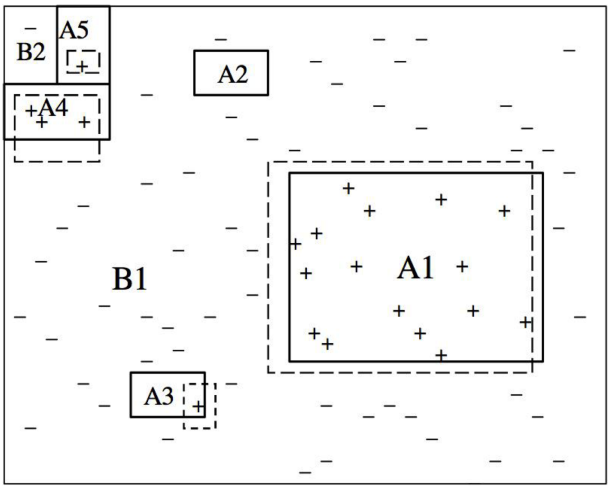

FIGURE 2.16 – Classes rares et cas rares [Wei04]

Définition 2.7.3. *[Gom07, HAP89] Un* ***Small Disjunct*** *est un ensemble d'instances ciblé par une règle qui couvre un très faible pourcentage d'instances du jeu de données.*

Le fait qu'un Small Disjunct ne couvre que peu d'instances fait que son taux d'erreur est généralement plus important que celui d'une règle couvrant un grand nombre d'instances.

2.7.1 Difficultés liées aux classes/cas rares

2.7.1.1 Rareté absolue et relative

Le concept de rareté dans un jeu de données est un concept général, que nous devons préciser en divisant celui-ci en deux parties : la rareté absolue et la rareté relative [Wei04].

Définition 2.7.4. *[Wei04] La* ***rareté absolue*** *est associée à un faible nombre d'instances de cas/classes rares au sens absolu du terme, c'est-à-dire sans comparer le nombre d'instances avec la (le) classe (cas) majoritaire (commun). Ce manque de données dans le jeu de données fait qu'il est très difficile de détecter des régularités pour les cas/classes rares.*

La Figure 2.17 présente les problèmes liés au manque absolu d'instances dans un jeu de données. La partie gauche de la Figure montre une région $A3$ ne comprenant qu'une seule instance positive tandis que la partie droite illustre une région $A3$ où de nombreuses instances positives sont présentes.
Les bornes de décision en phase d'apprentissage sont très différentes des bornes idéales. Dans l'exemple de gauche, le manque absolu de données fait que les bornes d'apprentissage ne correspondent pas du tout aux bornes idéales. A contrario, dans l'exemple de droite les deux bornes sont similaires.

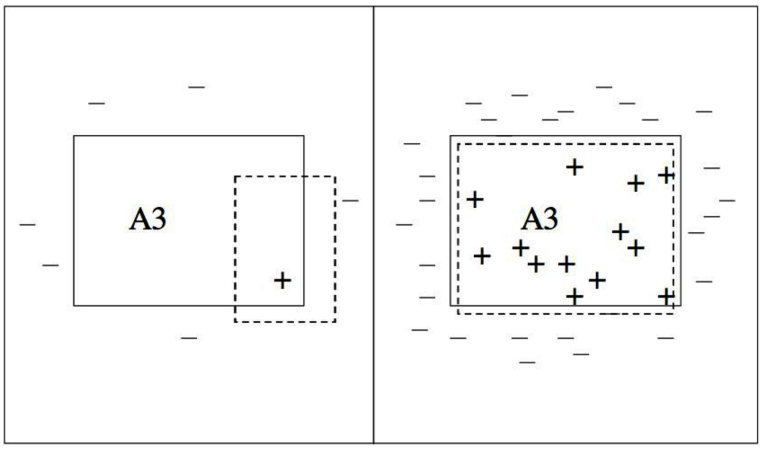

FIGURE 2.17 – Problème résultant d'un manque absolu de données [Wei04]

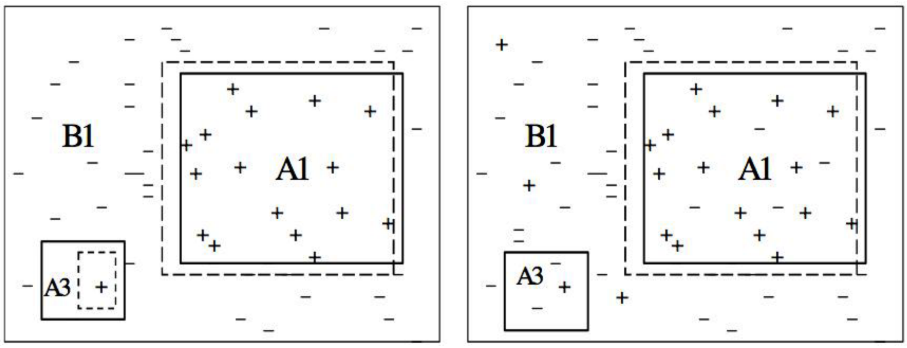

FIGURE 2.18 – jeu de données original (gauche) et jeu de données bruité (droite) [Wei04]

Définition 2.7.5. *La **rareté relative** est associée à des instances qui ne sont pas rares au sens absolu mais qui sont beaucoup **moins représentées** que des instances d'autres classes.*

Pour comprendre la différence entre la rareté absolue et relative, nous pouvons énoncer qu'un jeu de données de distribution (10 ; 150) sera lié au concept de **rareté absolue** tandis qu'un jeu de données de distribution (100 ; 1500) sera considéré comme étant lié à un problème de **rareté relative**. Néanmoins, la distinction entre rareté relative et absolue n'est pas clairement définie.

Dans ce travail, nous nous intéressons principalement à des jeux de données asymétriques ou par conséquent est présente une rareté relative.

2.7.1.2 Bruit

Définition 2.7.6. *Le **bruit** dans un jeu de données est l'ensemble des instances qui n'apportent aucune information intéressante dans la compréhension de celui-ci.*

Nous illustrons un exemple de bruit dans un jeu de données au moyen de la Figure 2.18.

Le jeu de données représenté par la partie droite de la Figure 2.18 est le jeu de données de la partie gauche de cette même figure auquel nous ajoutons du bruit. Dans cet exemple, le bruit est créé par le biais d'instances de classe négative ajoutées dans des régions majoritairement composées d'instances de classe positive (et inversement).

Ce bruit, considéré comme problème récurrent lors du processus d'apprentissage à partir de jeux de données symétriques a un impact plus important sur les classes/cas rares car peu d'instances bruitées suffisent pour fausser de façon

importante les bornes d'apprentissage.
Par exemple, pour la Figure 2.18, le classifieur qui doit créer des bornes d'apprentissage est incapable de créer celles-ci pour la région $A3$, le bruit étant trop important dans celle-ci (cas rare). A contrario, nous savons construire des bornes d'apprentissage pour la région $A1$ malgré l'ajout de bruit, $A1$ n'étant pas un cas rare.

2.7.2 Techniques inadaptées

2.7.2.1 Fragmentant les données

Les arbres de décision (Section 2.3) sont une méthode de type "diviser pour régner" et segmentent par conséquent le jeu de données en sous-ensembles de plus en plus réduits. Une fragmentation excessive du jeu de données favorise le surajustement (Section 2.3.5), d'où l'importance de réduire celle-ci en construisant des arbres de décision plus adaptés à ce type de problème ou en utilisant d'autres techniques.

2.7.2.2 Mesures

Afin de créer un classifieur et plus particulièrement un arbre de décision, nous utilisons des mesures permettant de construire celui-ci. En présence de jeu de données asymétriques, il convient de choisir parcimonieusement ces mesures afin de ne pas ignorer la classe minoritaire.
Nous allons donner des exemples de mesures inadaptées, certaines dédiées à la qualité des classifieurs, d'autres utilisées comme fonctions d'impureté ou encore comme règles d'affectation.

Certaines mesures d'**évaluation de la qualité** ne sont pas adaptées à l'asymétrie. La mesure "accuracy"(Formule 2.14) calcule le pourcentage d'instances correctement classées [Wei04]. En présence d'un jeu de données asymétrique, par exemple un jeu de données de distribution (10 ; 90), les performances sur la classe majoritaire seront en moyenne neuf fois supérieures aux performances du classifieur sur les instances de la classe minoritaire. L'accuracy n'est par conséquent pas une mesure adaptée à la classe minoritaire. Cette affirmation a été confirmée par une étude [WP03]. Celle-ci a montré que sur 26 jeux de données, le taux d'erreur de classification des règles de la classe minoritaire est en moyenne deux à trois fois plus important que le taux d'erreur de classification des règles de classe majoritaire. Les instances de classe minoritaire sont donc moins bien prédites que leurs homologues majoritaires.

Certaines **fonctions d'impureté** ne conviennent pas aux jeux de données asymétriques. Comme le montre la Figure 2.7, l'entropie et le Training Set Error sont des mesures dont la valeur maximale (voir propriété 2 de la Définition 2.3.2) correspond à la distribution uniforme. Cette propriété pose un problème qui peut être illustré par l'exemple suivant. Supposons un jeu de données destiné à construire un arbre de décision permettant de détecter un cancer. La distribution du jeu de données d'apprentissage est (10%, 90%). La création d'une feuille de distribution (50%, 50%) sera généralement rejetée car elle possède une valeur d'entropie ou de Training Set Error maximale alors que, pour un jeu de données asymétriques, cette distribution est intéressante (la moitié des individus ciblés par cette feuille ont le cancer, ce qui peut être considéré statistiquement intéressant par rapport à la distribution initiale).

Les règles d'affectation permettant d'assigner une classe à une feuille (Section 2.3.6) peuvent également parfois désavantager les classes minoritaires. En effet, la règle d'affectation la plus courante est d'assigner à une feuille la classe à laquelle appartient plus de la moitié des instances (pour un jeu de données ayant un attribut cible binaire). Or, comme pour l'exemple précédent, en présence d'un jeu de données asymétrique, nous pouvons assigner une classe à une feuille lorsque le pourcentage d'instances de cette classe est situé en dessous de 50% [Mar].

2.7.3 Techniques adaptées

2.7.3.1 Mesures

Dans cette partie, nous présentons quelques mesures qui ne discriminent pas la classe minoritaire, certaines dédiées à la qualité des classifieurs et d'autres utilisées comme fonctions d'impureté.

La mesure **ROC** (Section 2.5.1) est intéressante lorsque nous manipulons des jeux de données asymétriques [Faw06] car n'est pas sensible à la distribution. Si la proportion d'instances de classe positive ou de classe négative change, la courbe ROC ne change pas [Faw06].
La première ligne d'une matrice de confusion (Table 2.4) est la proportion d'instances positive dans le jeu de données alors que la deuxième indique la proportion d'instances négatives dans le jeu de données. Contrairement à des mesures telles que la précision et l'accuracy qui sont calculées à partir des **deux lignes** de la matrice de confusion, les mesures **TPR** et **FPR** utilisées par ROC ne dépendent respectivement que d'une seule ligne, ce qui fait que la mesure

ROC est une mesure adaptée aux jeux de données asymétriques.

Au niveau des fonctions d'impureté, plusieurs mesures ont été conçues dans le but de prendre spécifiquement en compte la classe minoritaire en transformant la propriété 2 de la Définition 2.3.2, telles que l'**entropie décentrée** ou l'**entropie asymétrique** [Mar, LLV07].

2.7.3.2 Choix du biais inductif

Lorsque nous devons apprendre à partir de jeux de données asymétriques, de nombreuses méthodes favorisent la généralisation au détriment de la spécialisation (Section 2.2.1) [Mar]. Pourtant, la généralisation n'est pas appropriée aux cas rares car celle-ci favorise les cas fréquents [Wei04]. Il est donc nécessaire de choisir un biais plus adapté aux cas rares.

2.7.3.3 Apprentissage sensible aux coûts

Une autre technique permettant d'accorder plus d'importance à la classe minoritaire est l'apprentissage sensible aux coûts. Dans celui-ci, nous considérons, comme dans la vie réelle que l'importance d'un **FP** (*détecter un cancer chez un individu sain*) est différent de l'importance d'un **FN** (*ne pas détecter le cancer d'un individu malade*).

Nous créons une matrice, semblable à une matrice de confusion mais qui contient les coûts de mauvaise classification [Mar]. Cette matrice est la Table 2.8.

		Prédiction	
		Négatif	Positif
Actuel	Négatif	TN = 0	FP = 1
	Positif	FN = 2	TP = 0

Table 2.8 – Matrice des coûts de mauvaise classification

La matrice de mauvaise classification 2.8 peut par exemple être celle d'un problème de détection de cancer à partir d'un jeu de données de patients. Détecter l'absence de maladie sur un patient sain et la présence d'un cancer chez un patient atteint correspondent logiquement à des coûts nuls (puisqu'il n'y a pas "mauvaise classification"). La matrice donne un coût d'une unité lorsqu'un cancer est détecté chez un patient sain (coût de faire des analyses inutiles et d'inquiéter le patient sans raison). Un coût de deux est donné lorsque le classifieur ne détecte pas le cancer d'un individu pourtant malade. Le but d'un

classifieur sensible au coût n'est pas de minimiser le nombre d'erreurs mais de minimiser le coût total.

La matrice de mauvaise classification intervient également dans l'affectation d'une conclusion à une feuille d'un arbre de décision [JBVW06]. Imaginons qu'une feuille de l'arbre ait une distribution de forme (+,-) (4,6). La conclusion de la feuille est généralement la classe négative étant donné le plus grand nombre d'instances négatives. Si nous utilisons une matrice de coût de mauvaise classification, telle que la matrice de la Table 2.8, le coût lié à une conclusion négative sera 4*coût FN = 4*2 = 8. Le coût lié à une conclusion positive sera 6*coût FP = 6*1 = 6. La conclusion minimisant le coût de la feuille est donc la conclusion positive.

2.7.3.4 Considérer uniquement la classe rare

Lorsque l'on essaye d'obtenir des règles de bonne qualité pour chaque classe du jeu de données, le résultat peut-être médiocre car en ne se focalisant pas sur une classe, la plus importante à nos yeux, il se peut que nous ne la considérions pas à sa juste valeur, que nous ne percevions pas son importance dans le problème. Pour contrer cela, une technique se focalise uniquement sur les instances de la classe minoritaire, comme présenté dans [JMG95].

D'autres outils ont été développé dans le but de n'apprendre que de la classe minoritaire. Nous pouvons par exemple citer **Brute** [RSE94], **Shrink** [KHM97] et **Ripper** [Coh95].

Nous pouvons également citer les méthodes de sampling, c'est-à-dire des méthodes qui ajoutent (*oversampling*) ou qui retirent (*undersampling*) des instances d'un jeu de données. Une méthode de sampling assez connue est Smote [CBHK02]. La présentation de ces différentes méthodes est faite dans [Wei04]. Des algorithmes génétiques donnent généralement de bons résultats sur des jeux de données asymétriques [Wei99] mais leur étude sort du cadre de ce travail.

3

Algorithme et Implémentation

Dans ce chapitre, nous exposons nos apports. Nous commençons par énoncer la **problématique** (Section 3.1) de façon plus précise. Après, nous décrivons les **modifications** apportées aux **concepts** importants de ce travail afin qu'ils soient à même d'appréhender la discrétisation (Section 3.3).
Ensuite, nous présentons notre **algorithme de base** (Section 3.4) de création d'arbres de décision sur lequel reposent les deux instances que nous avons développées (Sections 3.5 et 3.6). Cet algorithme, en plus de permettre la construction d'arbres de décision en utilisant la discrétisation locale, permet également leur élagage grâce à l'implémentation d'une technique de **post-élagage**.
La section suivante du présent chapitre présente le premier arbre de décision créé (Section 3.5). Celui-ci a pour critère de segmentation le **Training Set Error**. Par après, nous présentons le deuxième arbre de décision élaboré (Section 3.6). Celui-ci a été construit dans le but d'être plus adapté que le premier aux jeux de données asymétriques et utilise l'**entropie décentrée** comme critère de segmentation.

3.1 Asymétrie et attributs quantitatifs

Ce travail présente un algorithme de création d'arbres de décision conçu afin d'améliorer les performances de classification sur un type particulier de jeux de données [1].
Ces jeux de données possèdent **deux spécificités**. Premièrement, ils sont **asymétriques** (Section 2.7), ce qui signifie qu'une classe est largement moins représentée que l'autre classe [2]. Deuxièmement, les attributs indépendants de ce type de jeu de données sont **quantitatifs**.

De nombreux jeux de données sont asymétriques, tels que ceux étudiés dans [CS98, KHM98, WH98]. Dans la Section 2.7, nous avons énoncé un certain nombre de problèmes apparaissant lors de la classification de jeux de données contenant une classe minoritaire. La faible proportion d'instances de la classe minoritaire associée aux techniques et au biais inadaptés choisis par de nombreux classifieurs (biais favorisant la généralisation sur la spécialisation [Wei04]) fait que la classe minoritaire est généralement mal classée. Pourtant, ce sont très souvent ces instances qui sont les plus importantes pour les personnes étudiant les phénomènes représentés par ce type de jeu de données.
Dans certains cas, les attributs de ces jeux de données asymétriques sont **quantitatifs**. De nombreuses méthodes ne gèrent pas ce type d'attributs et ils sont souvent pré-discrétisés. Ceux-ci peuvent également directement être utilisés avec des algorithmes de classification tels que C4.5 (qui permet une discrétisation locale). Néanmoins, les techniques de pré-discrétisation ou encore la discrétisation locale de C4.5 n'ont pas été spécialement conçues pour la classe minoritaire (fonctions d'impureté inadéquates).

L'intérêt pour ce type de problème, outre le fait qu'il n'existe à ce jour pas de technique qui fasse entièrement l'unanimité, est qu'il faille généralement associer plusieurs méthodes venant de domaines différents afin d'obtenir des performances acceptables. La multiplicité des techniques ainsi que la difficulté de savoir lesquelles associer entre elles font que concevoir une solution pour ce type particulier de problème est assez complexe, tant à construire qu'à expérimenter.

De nombreuses méthodes ont été développées dans le but d'améliorer les performances de classification sur les jeux de données asymétriques. Il n'y a pas, à notre connaissance, une technique spécialement adaptée (conçue) aux

1. La présente section donne une vision plus détaillée du problème que la Section 2.1.3.
2. Notre travail se focalisant sur la classification binaire, l'attribut dépendant ne peut prendre que deux valeurs.

attributs quantitatifs présents dans les jeux de données asymétriques et intégrée dans un algorithme de création d'arbres de décision. Nous allons donc citer quelques techniques adaptées aux jeux de données asymétriques [Wei04].

L'algorithme **CN2**, algorithme appartenant au subgroup discovery [HCGdJ10] possède un biais plus spécifique et est donc plus adapté aux jeux de données asymétriques. Il donne de bons résultats sur les "small disjuncts" mais de moins bons sur les "large disjuncts". D'autres techniques cherchent à améliorer les performances en présence de jeux de données asymétriques. **Brute**, **Shrink** et **Ripper**[Wei04] se focalisent sur la classe minoritaire et ne sont pas aptes à classer des instances n'appartenant pas à celle-ci.
D'autres méthodes modifient la distribution du jeu de données afin que la distribution soit uniforme. Ces méthodes sont des méthodes de "sampling". Ce rééquilibrage du jeu de données se fait soit en ajoutant des instances de la classe minoritaire, soit en supprimant des instances de la classe majoritaire. La méthode de "sampling" la plus connue est **SMOTE**.
Un autre type d'algorithme est de plus en plus présent en data mining : l'algorithme génétique. Plusieurs études ont par ailleurs montré les qualités de l'algorithme génétique lorsque celui-ci fait face à des jeux de données asymétriques [Wei04].

Des algorithmes d'arbres de décision ont également été conçus pour les jeux de données asymétriques. Ainsi, [Mar] expérimente plusieurs fonctions d'impureté telles que l'entropie décentrée et l'entropie asymétrique afin de créer des arbres de décision à partir de jeux de données asymétriques aux attributs indépendants qualitatifs. De même, dans [LCCC10], une nouvelle mesure a été développée, la "Class Confidence Proportion". Celle-ci est associée à une méthode de post-élagage présentant de bons résultats par rapport à la classe minoritaire, le post-élagage au moyen du test exact de Fisher.

Notre solution se positionne donc dans l'ensemble des techniques construisant des arbres de décision adaptés aux jeux de données asymétriques. Néanmoins, elle se différencie de celles-ci dans le sens où elle discrétise de façon locale les attributs indépendants quantitatifs au moyen d'une méthode sensée convenir à la classe minoritaire.
Nous exposons notre solution de la manière suivante. Nous commençons par exposer l'algorithme de base de création d'arbres de décision implémenté et intégré au logiciel associé à ce travail (voir Annexe A.1). Nous présentons par

la suite les deux instances de cet algorithme qui ont été expérimentées [3]. Une instance (Section 3.5) intègre dans son processus de segmentation le "Training Set Error", mesure inadaptée aux jeux de données asymétriques. La deuxième instance (Section 3.6) intègre dans son processus de segmentation "l'entropie décentrée", mesure spécialement conçue pour les situations d'asymétrie et donnant de bons résultats [Mar, LLV07].

Afin de pouvoir présenter de manière claire les arbres de décision adaptés à ce type de jeux de données, nous devons mettre à jour le jeu de données de la Table 2.1. Nous avons dans ce but créé un jeu de données asymétrique [4] ne contenant que des attributs indépendants quantitatifs.
Ce jeu de données est présenté par la Table 3.1. Il est similaire à celui de la Table 2.1 dans le sens où notre objectif est d'expliquer un attribut dépendant "Dossier Approuvé" en fonction d'un ensemble d'attributs indépendants. Les attributs de ce jeu de données sont les suivants :

- **indépendants** :
  - **Salaire** : le salaire mensuel d'un individu, attribut quantitatif continu
  - **Age** : l'âge de l'individu, attribut quantitatif discret
  - **Remboursement** : la somme des prêts que doit rembourser un individu mensuellement, attribut quantitatif continu
  - **P. charge** : le nombre de personnes à charges d'un individu, attribut quantitatif discret
- **dépendant** :
  - **Dossier approuvé** : attribut cible, peut prendre *oui* ou *non* comme valeur

3.2 Choix effectués

3.2.1 Arbre de décision

Lorsque nous avons dû choisir l'algorithme de classification le plus adapté au problème donné, nous avons sélectionné, après l'étude de plusieurs techniques de classification, l'une des méthodes les plus connues : les arbres de décision.
En plus des avantages listés dans la Section 2.3.8, les arbres de décision peuvent être **facilement appréhendés** par une majorité de personnes, même des non-

3. Etant donné que nous changeons plusieurs paramètres pour chaque instance comme par exemple l'arité, plus de deux "instances" sont expérimentées.

4. Exemple uniquement pédagogique, un jeu de données ne contenant que 15 instances ne peut pas réellement être considéré comme asymétrique.

Identifiant	Salaire	Age	Remboursement	P. charge	Dossier approuvé ?
1	1202	21	313.2	1	non
2	1373	32	128.4	4	non
3	2100	44	703.7	3	oui
4	1505	24	405.9	2	non
5	1949	48	102.8	0	oui
6	1734	26	850.3	1	non
7	1621	35	307.8	5	non
8	3100	57	1003.4	0	oui
9	2756	29	840.7	2	oui
10	1173	28	258.9	3	non
11	2512	47	628.4	0	oui
12	2839	41	986.0	1	non
13	3004	34	1248.5	5	non
14	2076	34	512.8	4	oui
15	2776	28	730.1	3	non

TABLE 3.1 – Exemple de jeu de données où les attributs indépendants sont quantitatifs

spécialistes du data mining. Le caractère **exploratoire** de cette méthode permet de renforcer l'aspect pédagogique de celle-ci [5]. De plus, nous pouvons facilement dériver des règles à partir de ceux-ci, ce qui permet de donner un regard nouveau sur le classifieur.
Les arbres de décision utilisés dans ce travail sont également une base solide à partir de laquelle nous pouvons faire évoluer notre modèle vers des structures plus complexes telles que le *bagging* [Bre96].

Même si les arbres de décision ne sont initialement pas une technique spécialement conçue pour les jeux de données asymétriques (Section 2.7), nous avons construit ceux-ci en utilisant des méthodes qui ont fait leurs preuves face à ce genre de données (élagage avec test exact de Fisher, entropie décentrée,...) ainsi qu'une technique qui, selon nous, peut donner de bons résultats grâce à la découverte de zones plus précises dans l'espace des instances (discrétisation locale).

3.2.2 Discrétisation locale

Comme nous venons de l'énoncer, notre travail intègre plusieurs techniques adaptées aux jeux de données asymétriques. Nous avons fait le choix d'ajouter

5. Les classifieurs construits sont souvent étudiés par des personnes expertes du domaine dans lequel se situent les jeux de données et non par des spécialistes du data mining. Un modèle aisé à comprendre et à représenter, tel qu'un arbre de décision (qui peut de surcroît être transformé en un ensemble de règles de décision) est un atout non-négligeable.

également une technique n'ayant pas spécialement été expérimentée pour les jeux de données asymétriques mais qui peut potentiellement être adaptée à ceux-ci : la **discrétisation locale**.
Cette technique est utilisée depuis un certain temps par des algorithmes tels que C4.5. Dans ce travail, nous utilisons la discrétisation locale car nous pensons qu'elle peut donner de bons résultats lorsqu'elle est confrontée à des jeux de données asymétriques. En effet, en ciblant de manière précise des sous-ensembles d'instances (en discrétisant plusieurs fois un même attribut), nous discrétisons de manière locale afin d'être assez spécifiques par rapport à la classe minoritaire. N'ayant pas trouvé de critère d'arrêt adapté à l'asymétrie et nos tests ayant montré que le MDL ne convenait pas à la discrétisation locale, nous avons fait le choix de l'utilisation d'une arité fixe (pour l'instance 2 de l'algorithme de base, la première instance utilisant le Training Set Error et l'arité bornée).

3.2.3 Post-élagage

Une discrétisation appliquée de nombreuses fois (locale) peut favoriser le **surajustement**. Le surajustement peut également, pour l'instance 2 de l'algorithme de base, être favorisé par l'arité fixe (nous discrétisons les noeuds jusqu'à ce que le nombre d'instances présentes dans ceux-ci soit en dessous d'un certain seuil). Afin de limiter ce surajustement sans pour autant "négliger" la classe minoritaire, nous avons opté pour une méthode de **post-élagage**.
Les méthodes de post-élagage fournissent généralement de meilleurs résultats que les méthodes de pré-élagage (Section 2.3.5). Nous avons décidé d'intégrer la méthode de post-élagage utilisant le test exact de Fisher, méthode conçue par [LCCC10] et surpassant, d'après les expérimentations de cet article, la méthode d'élagage de C4.5.

3.3 Modifications conceptuelles

Avant d'exposer notre algorithme de classification, nous devons modifier deux concepts largement détaillés dans le chapitre précédent : les **arbres de décision** (Section 2.3) et les **règles** (Section 2.4).

3.3.1 Arbre et discrétisation locale

Lorsque nous avons présenté les arbres de décision, ceux-ci étaient construits à partir de jeux de données dont les attributs indépendants étaient qualitatifs. Lors de l'étape de segmentation de l'arbre de décision (Section 2.3.4), l'ensemble d'instances du noeud courant était simplement partitionné suivant les valeurs de l'attribut sélectionné.
A présent et étant donné notre volonté de construire des arbres de décision à partir de jeux de données contenant des attributs quantitatifs et de discrétiser de façon *locale* (Section 2.6), nous devons enrichir le processus de segmentation qualitatif défini dans la Section 2.3.1 afin qu'il devienne un processus de segmentation quantitatif.

Le processus de segmentation quantitatif (en présence d'attributs quantitatifs) comprend par conséquent trois étapes :

1. **Discrétisation** de tous les attributs indépendants du jeu de données.
2. Création d'une partition pour chaque attribut **discrétisé** du jeu de données. Cette partition est ensuite évaluée au moyen d'une fonction d'impureté.
3. Création d'un noeud fils à partir de chaque sous-ensemble de la partition correspondant à l'attribut ayant été choisi (celui qui produit la meilleure valeur à la fonction d'impureté).

La Figure 3.1 illustre les similitudes entre les deux processus de discrétisation [6]. Nous pouvons voir que les deux dernières étapes sont **identiques** pour les deux processus, ce qui est logique étant donné que l'attribut quantitatif est discrétisé. Celui-ci devient donc qualitatif et peut par conséquent être traité par un processus classique de segmentation qualitatif.

Afin de donner une meilleure compréhension du processus de segmentation quantitatif, nous illustrons celui-ci au moyen d'un exemple. Cet exemple se

6. le processus de segmentation qualitatif et le processus de segmentation quantitatif.

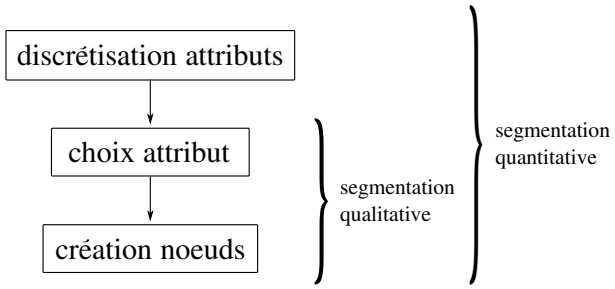

FIGURE 3.1 – Segmentation en présence d'attributs qualitatifs et quantitatifs

déroule en trois étapes suivant le processus de segmentation en présence d'attributs quantitatifs défini ci-dessus.

Etape 1 :
Nous discrétisons suivant une méthode donnée chacun des attributs suivants :

"Salaire", "Age", "Remboursement" et "P. charge" afin de savoir quel attribut produit la meilleure discrétisation (l'attribut qui a la meilleure valeur pour la fonction d'impureté donnée).

Etape 2 :
Après avoir évalué chacun des attributs, il ressort de ce test que c'est l'attribut

"Salaire" qui est le plus à même de segmenter le noeud racine. Le noeud racine de notre arbre est maintenant représenté par la Figure 3.2.

Salaire
6 - 9

FIGURE 3.2 – Noeud racine d'un arbre de décision construit à partir du jeu de données de la Table 3.1

La discrétisation de l'attribut "Salaire" suivant une méthode donnée discrétise "Salaire" en trois intervalles de valeur :

- $[0; 1500]$
- $]1500; 2500]$
- $]2500; \infty[$

Etape 3 :
Cette étape produit les noeuds fils à partir du noeud de la Figure 3.2. Chaque

noeud fils contient un ensemble d'instances ayant une valeur d'attribut "Salaire" appartenant à l'un des trois intervalles suivants : $[0; 1500]$, $]1500; 2500]$ et $]2500; \infty]$. Le résultat de l'étape de segmentation est visible à la Figure 3.3.

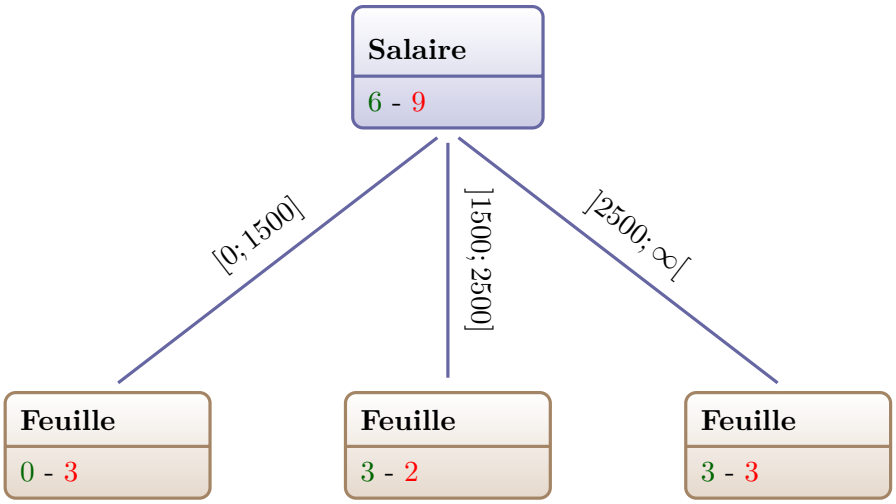

FIGURE 3.3 – Création des noeuds fils suivant les valeurs de l'attribut discrétisé

3.3.2 Règles et discrétisation

Après avoir expliqué les répercussions de la discrétisation locale sur les arbres de décision, nous devons expliquer celles qu'elles ont sur les règles (Section 2.4). Pour cela, nous allons modifier la définition de règle (définition originale : Définition 2.4.1).

La notion de valeur d'un couple (attribut, valeur) présent dans la partie antécédente de la règle doit être modifiée. En effet, celle-ci est toujours une valeur d'attribut qualitatif mais n'est ni nominale ni ordinale mais est un intervalle. Les intervalles résultant de la discrétisation peuvent être de plusieurs types, comme illustré par la Figure 3.4 : les intervalles possédant **deux bornes** finies (*type 1*) et les intervalles ne possédant qu'**une borne** finie (*type 2*).

Soit a et b, deux valeurs quantitatives.
Un intervalle possédant deux bornes finies peut être :

- **ouvert** : $]a; b[$
- **fermé** : $[a; b]$
- **semi-ouvert** : $]a; b]$ ou $[a; b[$

Dans le cas d'un intervalle possédant une borne finie, celle-ci peut être :

– **borne supérieure** : $]\infty; a]$ ou $]\infty; a[$
– **borne inférieure** : $[a; \infty[$ ou $]a; \infty[$

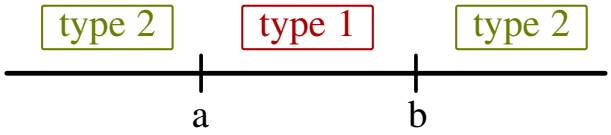

FIGURE 3.4 – Illustration de différents types de bornes

Après avoir introduit brièvement la notion d'intervalle, nous pouvons donner une nouvelle définition du concept de règle.

Tout comme la Définition 2.4.1, une règle adaptée à la discrétisation comporte toujours deux parties, la **partie antécédente** et la **partie conséquente**.

Définition 3.3.1. *Une règle i prend la forme suivante [HCGdJ10] :*

$$\mathbf{R_i} : Condition \rightarrow Attribut_{Cible} \tag{3.1}$$

où

– ***Conditions**, la partie **antécédente** de l'implication est une conjonction conditions, c'est-à-dire de couples de type $(attribut, valeur)$ [GB10]. La valeur du couple contenant un attribut discrétisé (anciennement quantitatif) est à présent un **intervalle**.*
Une condition

$$(attribut_1, valeur_1) \wedge (attribut_2, valeur_2) \wedge \ldots (attribute_{n-1}, valeur_{n-1})$$

*est **Vraie** pour une instance quelconque i lorsque l'ensemble des valeurs des attributs $\{attribut_1, attribut_2, \ldots attribut_{n-1}\}$ de i appartiennent aux valeurs (intervalles) des attributs correspondants de la condition.*

– **Attribut$_{\text{Cible}}$**, *la partie **conséquente** de l'implication représente la valeur de l'attribut cible attribuée à la règle [GB10].*

Un exemple de règle contenant des attributs ayant été au préalable discrétisés est la Règle 3.2. Pour qu'une instance appartienne à cette règle, il faut que la valeur de l'attribut salaire soit comprise entre 1500 et 3000 euros et que son remboursement soit compris entre 0 et 300 euros. Si c'est le cas, le classifieur lui attribuera la variable d'attribut cible "Dossier Approuvé = oui".

$$Salaire \in]1500; 3000] \wedge Remboursement \in]0; 300] \rightarrow DossierApprouvé = oui \tag{3.2}$$

3.4 Algorithme de base

Dans cette partie, nous décrivons l'algorithme que nous avons conçu et implémenté. Celui-ci est un algorithme de construction d'arbres de décision (et de règles dérivées de ceux-ci) de base [7] qui possède deux instances expliquées dans les parties ultérieures (Sections 3.5 et 3.6). Cet algorithme est similaire à un algorithme tel que C4.5 dans la mesure où ils permettent tous les deux une discrétisation locale (voir 2.3.7).

Dans la suite de cette section, nous expliquons l'algorithme de base avant de détailler ses points clés.

L'algorithme de base (Algorithme 2) est composé de **trois parties** : une partie construisant l'arbre de décision (Algorithme "Create the decision tree", voir Section 3.4.1), une partie élaguant l'arbre produit (Algorithme "Prune the decision tree", Section 3.4.2) et une partie construisant les règles (Algorithme "Create the rules", Section 3.4.3).

Algorithme 2: Basic Algorithm

```
Create the decision tree ;
Prune the decision tree ;
Create the rules ;
```

3.4.1 Création de l'arbre de décision

Cet algorithme "Create the decision tree" crée l'arbre de décision. L'Algorithme 3 est le pseudo-code de celui-ci. Cet arbre de décision est créé en "largeur d'abord" et utilise la segmentation quantitative (Section 3.3.1). Celui-ci ne s'arrête que lorsque le critère d'arrêt est atteint.
La paramètre arité de cet algorithme est introduit manuellement et sera utilisé pour l'arité bornée (instance 1) ou l'arité fixe (instance 2).

3.4.2 Elagage de l'arbre de décision

Afin que les arbres produits aient de bonnes performances par rapport aux jeux de données de test, nous devons vérifier que ceux-ci ont des capacités de

7. Celui-ci n'est adapté, comme dit précédemment, qu'aux jeux de données possédant **deux classes**.

Algorithme 3: *CreateDecisionTree*(*Instances*, $Target_{Attribute}$, *Attribute*)

Data : *Instances* is the dataset, $Target_{Attribute}$ is the class value, *Attributes* is the Attribute list, *Method* is the discretization method, *Arity* is the arity of the discretization and *Threshold* is the minimum size of *Instances*
input : *Instances*, $Target_{Attribute}$, *Attribute*, *Method*, *Arity*
output : tree

```
create a Root node for the tree;
if all Instances have the same class value then
    return single-node tree Root with label =Target_Attribute of Instances;
if Attributes is empty then
    return single-node tree Root, with label = most common value of
    Target_Attribute in Instances;
begin
    discretize all the attributes of Instances of the node i with Method ;
    A ← the attribute which is the best following Method;
    foreach possible discretized value v_i of A do
        Add a new tree branch below Root corresponding to the test
        A = v_i;
        Let Instances_{v_i} be the subset of Instances that have value v_i for
        A;
        if Instances_{v_i} is empty OR Instances_{v_i} < threshold then
            below this new branch add a leaf node with label = most
            common value of Target_Attribute in Instances;
        else
            below this new branch add the subtree
            CreateDecisionTree(Instances_{v_i}, Target_Attribute,
            Attributes);
return Root;
```

généralisation suffisantes (qu'ils ne sont pas *surajustés*). Pour cela, nous utilisons une méthode de **post-élagage** de l'arbre de décision (Section 2.3.5).

Cette méthode élague l'arbre de décision suivant un critère particulier, le *test exact de Fisher* (Section 3.4.2.1), un test d'hypothèses assez répandu. Plusieurs études ont montré que celui-ci est adapté aux jeux de données asymétriques [LCCC10] et qu'il surpasse la méthode d'élagage [Qui93] utilisée par C4.5 (Section 2.3.7).

3.4.2.1 Test Exact de Fisher

Définition 3.4.1. *Un **test d'hypothèses** est une procédure basée sur l'observation d'un ou plusieurs échantillons permettant de faire un choix entre deux hypothèses formulées.*

Les deux hypothèses sont les suivantes : l'**hypothèse nulle** (H_0) est celle que l'on cherche à réfuter et l'**hypothèse alternative** (H_1) est celle que l'on cherche à démontrer [tes]. L'hypothèse nulle est soumise au test et supposée vraie [Gen09].

Deux types d'erreurs peuvent survenir lors d'un test d'hypothèses :
- l'erreur de première espèce : l'erreur de rejeter H_0 alors qu'elle est vraie
- l'erreur de seconde espèce : l'erreur d'accepter H_0 alors qu'elle est fausse

Lorsque l'erreur de première espèce est en-dessous d'un certain seuil, nous pouvons rejeter l'hypothèse H_0 [8] [tes].

Un test exact de Fisher est un test d'hypothèses adapté aux petits échantillons [Gen09] et qui mesure la corrélation entre deux (ou plusieurs) variables. Pour cela, nous construisons la matrice de contingence de la Table 3.2 afin d'étudier le **lien hypothétique entre deux variables**, par exemple une propriété P et une relation R. La propriété P peut être soit présente (P_1) soit absente (P_2). La propriété R peut être soit présente (R_1) soit absente (R_2) [ide]. Les éléments a, b, c, d sont les nombres d'instances appartenant respectivement à chaque cas (exemple : a instances ont une propriété et une relation présentes). Les sommes ($a + b$, $c + d$, $a + c$, $b + d$) sont appelées les sommes marginales de la matrice de contingence. n est le nombre d'instances de la matrice de contingence.

8. Le seuil de signification est fixé à l'avance et vaut généralement 0.05.

L'indépendance statistique étant plus aisée à vérifier que la dépendance statistique, nous ne devons pas vérifier qu'un résultat R dépende d'une propriété P (hypothèse H_1), nous devons montrer qu'il est improbable que R ne dépende pas de P (hypothèse H_0) [ide].

		Relation		
		R1	R2	
Propriété	P1	a	b	**a+b**
	P2	c	d	**c+d**
		a+c	**b+d**	**n**

TABLE 3.2 – Matrice de contingence utilisée pour le test exact de Fisher

Pour calculer le seuil de signification du test exact de Fisher, nous commençons par calculer la probabilité d'avoir la matrice de contingence 3.2. Cette probabilité est définie par l'Equation 3.3.

$$p(a,b,c,d) = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{n!a!b!c!d!} \tag{3.3}$$

A partir de cette probabilité, nous pouvons calculer le test exact de Fisher qui est la probabilité d'obtenir cette matrice de contingence ou une **plus extrême**. La notion de plus d'extrême étant définie comme une matrice de contingence où la différence de proportion respectivement entre la présence ou l'absence de la propriété P suivant la présence ou l'absence de la relation R est encore plus grande, les sommes marginales étant inchangées. La probabilité d'avoir une matrice de contingence égale ou plus extrême à la Table 3.2 est égale à la Formule de Fisher 3.4.

$$Fisher(a,b,c,d) = \sum_{i=0}^{min(b,c)} \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{n!(a+i)!(b-i)!(c-i)!(d+i)!} \tag{3.4}$$

Afin d'illustrer ces formules, nous donnons un exemple de test exact de Fisher [BW08].
Supposons qu'un laboratoire médical veuille étudier le lien hypothétique entre l'usage de drogues et les arrêts cardiaques. Après avoir réalisé une étude sur 20 patients, ceux-ci obtiennent les résultats suivants, exprimés par la matrice de contingence de la Table 3.3.

Pour cet exemple, l'hypothèse nulle H_0 est la suivante : "le risque de mort suite à un arrêt cardiaque n'est pas plus fréquent chez les consommateurs de drogues que chez les autres".
L'hypothèse alternative H_1 est l'hypothèse contraire à l'hypothèse nulle expri-

		Mort d'arrêt cardiaque		
		oui	non	
Usage de drogues	oui	7	2	**9**
	non	5	6	**11**
		12	**8**	**20**

TABLE 3.3 – Matrice de contingence de l'étude sur le lien éventuel entre l'usage de drogues et les arrêts cardiaques

mant le fait que "le risque de mort suite à un arrêt cardiaque est plus important chez les personnes qui consomment de la drogue que chez les autres".

Le calcul de la probabilité (suivant la Formule 3.3) d'obtenir la matrice de contingence 3.3 est

$$p(7,2,5,6) = \frac{(7+2)!(5+6)!(7+5)!(2+6)!}{20!7!2!5!6!} = 0.132$$

Afin d'obtenir la valeur du test exact de Fisher, nous devons à présent calculer les probabilités d'occurrence des matrices de contingence plus extrêmes comme par exemple la matrice de contingence de la Table 3.4.

		Mort d'arrêt cardiaque		
		oui	non	
Usage de drogues	oui	8	1	**9**
	non	4	7	**11**
		12	**8**	**20**

TABLE 3.4 – Matrice de contingence **plus extrême** que la matrice de contingence de la Table 3.3

La Formule 3.4 permet de calculer la valeur du test exact de Fisher de manière directe.

$$\begin{aligned} p(7,2,5,6) &= \sum_{i=0}^{min(2,5)} \frac{(7+2)!(5+6)!(7+5)!(2+6)!}{20!(7+i)!(2-i)!(5-i)!(6+i)!} \\ &= \frac{(7+2)!(5+6)!(7+5)!(2+6)!}{20!7!2!5!6!} + \frac{(7+2)!(5+6)!(7+5)!(2+6)!}{20!8!1!4!7!} \\ &+ \frac{(7+2)!(5+6)!(7+5)!(2+6)!}{20!9!0!3!8!} \\ &= 0.157 \end{aligned}$$

Le seuil de signification du test exact de Fisher pour cet exemple est de

0.157, il est plus grand que la valeur 0.05 [9]. Nous pouvons donc dire que nous ne savons pas rejeter l'hypothèse H_0 entre les deux variables.

3.4.2.2 Algorithme

Après avoir expliqué le test exact de Fisher, nous pouvons nous concentrer sur l'algorithme l'utilisant (Algorithme "PruneTheDecisionTree"). Cet algorithme de post-élagage calcule le test exact de Fisher pour chaque noeud de l'arbre afin de savoir quels noeuds conserver. Le test exact de Fisher calcule donc la signification de chaque branche de l'arbre [LCCC10]. Une branche d'un arbre est représentée par la partie antécédente d'une règle, branche menant à un noeud (et non à une feuille de l'arbre). Nous mettons à jour les notations d'une règle afin d'intégrer le concept de branche dans la Formule 3.5.

$$X \rightarrow y \tag{3.5}$$

où X est une branche de l'arbre et y une classe.

Pour la règle 3.5, nous allons calculer la probabilité d'avoir la matrice de contingence 3.5 où X et y sont au moins aussi positivement associés. Chaque branche statistiquement non-significative est supprimée [LCCC10].

		Antécédent		
		X	$\neg X$	
Classe	y	a	b	**a+b**
	$\neg y$	c	d	**c+d**
		a+c	**b+d**	**n**

TABLE 3.5 – Matrice de contingence du test exact de Fisher utilisée pour élaguer un arbre de décision

Le **fonctionnement de l'algorithme de post-élagage** peut être résumé en deux étapes :

1. Chaque noeud de l'arbre est examiné en partant depuis les feuilles de celui-ci (processus *bottom-up*). Un noeud est marqué "élagable" si et seulement si tous ses descendants sont non-significatifs.
2. L'arbre est à présent parcouru en commençant par la racine. Un noeud de l'arbre devient une feuille si son statut est "élagable" (processus top-down).

9. seuil de signification conventionnel

L'algorithme général du post-élagage au moyen du test exact de Fisher correspond à l'Algorithme 4 ("Prune"). Celui-ci contient deux "sous-algorithmes". L'Algorithme 5 ("setPruneable") vérifie la signification statistique de chaque règle possible (branche dans l'arbre) et garantit que chaque règle significative non significative est "élagable" (étape 1 du fonctionnement ci-dessus). L'Algorithme 6 ("pruneByStatus") élague toutes les règles non-significatives (étape 2 du fonctionnement ci-dessus).

Algorithme 4: *Prune*($decisionTree, pVT$) [LCCC10]

```
Data : decision tree, p-Value Threshold
input  : Unpruned decision tree DT, p-value threshold (pVT)
output : Pruned DT
foreach Leaf_i do
    if Leaf_i.parent is not the Root of DT then
        Leaf_i.parent.pruneable = true;
        setPruneable(Leaf_i.parent, pVT);
foreach child(i) of the root do
    if child(i) is not a leaf then
        if child(i).pruneable == true then
            set child(i) to be a leaf;
        else
            pruneByStatus(child(i));
```

3.4.3 Création des règles

L' Algorithme "Create the rules" est la troisième partie de l'algorithme de base et permet de construire des règles à partir de l'arbre de décision. Cet algorithme s'inspire de l'algorithme décrit à la Section 2.4.2. Après la construction des règles, nous nettoyons celles-ci en supprimant par exemples les couples (attribut,valeur) ayant un attribut identique et dont l'intervalle (la valeur) de l'un est strictement inclus dans l'intervalle (la valeur) de l'autre.
Par exemple, une règle dont l'antécédent est composé de trois couples est représentée par 3.6.

Algorithme 5: *setPruneable*$(Node, pVT)$ [LCCC10]

Data : Node, p-Value Threshold
input : A branch node $Node$, p-value threshold (pVT)
output : tree

```
foreach child(i) which is a Node do
    if child(i).pruneable == false then
        Node.pruneable = false;
if Node.pruneable == true then
    Calculate the p-value of this node : Node.pValue;
    if Node.pValue < pVT then
        Node.pruneable = false;
if Node.parent is not the Root of the full tree then
    Node.parent.pruneable = true;
    setPruneable(Node.parent, pVT);
```

Algorithme 6: *pruneByStatus*$(Node)$ [LCCC10]

Data : Node, p-Value Threshold
input : A branch represented by its top node Node
output : tree

```
if Node.pruneable == true then
    set Node as a leaf;
else
    foreach child(i) of Node do
        if child(i) is not a leaf then
            pruneByStatus(child(i));
```

$$\begin{aligned} & Salaire \in]1500; 3000] \wedge Remboursement \in]0; 300] \\ \wedge \quad & Salaire \in]1500; 1750] \rightarrow DossierApprouvé = oui \end{aligned} \tag{3.6}$$

Dans la règle 3.6, nous enlevons le couple (Salaire ;]1500 ;3000]) car l'arbre de décision a réutilisé ultérieurement l'attribut "Salaire" afin de construire un couple plus "précis", le couple (Salaire,]1500 ; 1750]). Supprimer certains couples permet d'améliorer la lisibilité de la règle et ainsi de faciliter sa compréhension.

3.5 Arbre et TSE

Ce premier arbre de décision est créé à partir de l'algorithme de base et utilise comme mesure de segmentation quantitative le Training Set Error (Section 2.3.4).

3.5.1 Algorithme de discrétisation

Cette mesure étant très populaire et l'une des premières conçues pour discrétiser, de nombreux algorithmes de discrétisation utilisant celle-ci ont été créés. Nous utilisons la version considérée comme étant la plus performante en terme de complexité et détaillée dans [Rou01, ER02].

Cette version est une version de type "arité bornée". Afin d'obtenir de bonnes performances, celle-ci utilise la *programmation dynamique* [Buh] afin de ne pas recalculer l'ensemble des valeurs pour chaque arité. Nous ne détaillerons pas cet algorithme dans ce travail et nous laissons au lecteur le soin d'aller consulter les articles de référence à ce sujet [Rou01, ER02].
Nous qualifions cet algorithme de discrétisation au moyen de la taxonomie présentée à la Section 2.6.3 comme étant un algorithme étant **supervisé**, **hiérarchique**, **top-down** et **paramétrique**.

3.5.2 Arbre de décision et TSE

Cette première instance de l'algorithme de base de création d'arbres de décision intègre l'algorithme de discrétisation de [Rou01, ER02]. Celui-ci utilisant l'arité bornée, nous devons spécifier celle-ci avant la création d'un arbre de décision. L'arité de l'arbre de décision n'est par conséquent pas déterminée de manière "dynamique" comme lors de l'utilisation d'une méthode de discrétisation associée à un critère d'arrêt tel que le principe de Description de Longueur Minimale [FI93].

L'algorithme de création d'arbres de décision intégrant le processus de segmentation quantitatif utilisant le Training Set Error ne stoppe la segmentation du noeud courant que dans deux cas : lorsqu'il ne sait plus discrétiser celui-ci ou lorsque le cardinal de l'ensemble d'instances présent dans celui-ci est en-dessous d'un certain seuil.
Cette méthode n'étant pas considérée comme une méthode ayant de bons résultats sur des jeux de données asymétriques, nous détaillons le choix de son utilisation dans la section suivante.

3.5.3 Choix

Nous avons choisi cette méthode de discrétisation car elle est l'une des plus anciennes ainsi que l'une des plus populaires [Rou01, ER02]. Le Training Set Error n'étant pas particulièrement adapté aux jeux de données asymétriques [Rou01], nous avons choisi d'utiliser celle-ci afin de pouvoir comparer les gains que nous pourrions obtenir lors de l'utilisation d'une mesure plus adaptée à ce type de jeu de données (voir entropie décentrée, Section 3.6.1).
Ce type de méthode a également été adopté car nous voulions tester la différence de résultats obtenus lors de la création d'arbres n-aires (C4.5 ne permet qu'une discrétisation binaire) afin de tester les affirmations effectuées dans [Rou01]. Ces dernières portent sur le fait que la production d'arbres n-aires peut "*limiter la profondeur de l'arbre de décision en comparaison avec des arbres binaires et éventuellement donner de meilleures performances*".

La dernière raison pour laquelle nous avons opté pour le Training Set Error est notre volonté de développer une méthode de discrétisation sensible aux coûts [BV98]. Nous n'avons malheureusement pas pu intégrer celle-ci dans ce travail mais le choix du Training Set Error fait que son intégration future ne devrait pas poser problème.

3.6 Arbre et entropie décentrée

Cette partie explique le fonctionnement de la seconde instance de l'algorithme de base de création d'arbres de décision employant d'une méthode de discrétisation utilisant une forme d'entropie adaptée aux jeux de données asymétriques, l'**entropie décentrée**.

3.6.1 Entropie décentrée

L'entropie de Shannon (Formule 2.7) est une mesure largement utilisée pour discrétiser [FI93, JBVW06]. Néanmoins, celle-ci possède un inconvénient non négligeable : elle n'est pas adaptée aux jeux de données asymétriques. En effet, une des propriétés d'une fonction d'impureté classique (voir Propriété 2, Définition 2.3.2) exprime le fait que l'entropie de Shannon est maximale lorque la distribution est uniforme, c'est-à-dire lorsque chaque classe d'un ensemble d'instances contient le même nombre d'instances. Cette situation de distribution uniforme est atteinte lorsque le $d(C_t, T)$ de la Figure 3.5 de l'entropie de Shannon vaut 0.5, ce qui veut dire que la valeur est maximale lorsque nous présupposons que nous avons affaire à un jeu de données **symétrique**.

Imaginons que nous devions créer un classifieur pour un jeu de données asymétrique de personnes atteintes d'un cancer contenant une distribution (cancer, sain) de $(0.1, 0.9)$. Si nous utilisons l'entropie de Shannon lors de la discrétisation, celle-ci rejettera très probablement un ensemble d'instances de distribution $(0.5, 0.5)$ alors que, dans le cadre d'un jeu de données asymétriques, découvrir une règle ciblant des personnes dont la moitié est atteinte du cancer est un résultat potentiellement intéressant. Il signifie qu'une règle ciblant huit individus d'un jeu de données asymétrique (contenant des malades atteints du cancer) contient quatre malades et quatre individus sains. Une règle arrivant à cibler des instances où la moitié est positive est intéressante dans le cadre de jeux de données asymétriques.

Pour tenter de corriger cet inconvénient et d'adapter le la propriété 2 de la Définition 2.3.2 en fonction de la distribution initiale du jeu de données, plusieurs mesures ont été créées, telles que l'**entropie décentrée** et l'**entropie asymétrique**.

Nous utilisons dans ce travail l'**entropie décentrée** [LLV07] que nous avons insérée dans un processus de discrétisation utilisant initialement l'entropie de Shannon. Cette mesure permet à l'utilisateur de spécifier la distribution qui rend

la valeur de l'entropie décentrée maximale [10]. Celle-ci peut-être la distribution du jeu de données initial ou une distribution quelconque fixée par l'utilisateur. Elle n'est donc pas maximale lorsque $p = 0.5$ [11] mais lorsque $p = \theta$ où θ est la distribution introduite par l'utilisateur (généralement distribution du jeu de données initial). p est un des éléments de la distribution de l'ensemble des instances dont nous devons calculer l'entropie décentrée. Afin que la valeur d'entropie décentrée soit maximale lorsque $p = \theta$, nous devons transformer la distribution $(1-p, p)$ en $(1-\pi, \pi)$, où :

$$\pi = \frac{p}{2\theta} \; si \; 0 \leq p \leq \theta \tag{3.7}$$

$$\pi = \frac{p+1-2\theta}{2(1-\theta)} \; si \; \theta \leq p \leq 1 \tag{3.8}$$

A partir du π, nous pouvons définir l'entropie décentrée $H_\theta(p)$ comme étant l'entropie de distribution $(1-\pi, \pi)$:

$$H_\theta(p) = -\pi \log_2 \pi - (1-\pi) \log_2 (1-\pi) \tag{3.9}$$

L'ACE (voir Formule 2.8), lors de l'utilisation de l'entropie décentrée est calculée en remplaçant l'entropie par l'entropie décentrée.

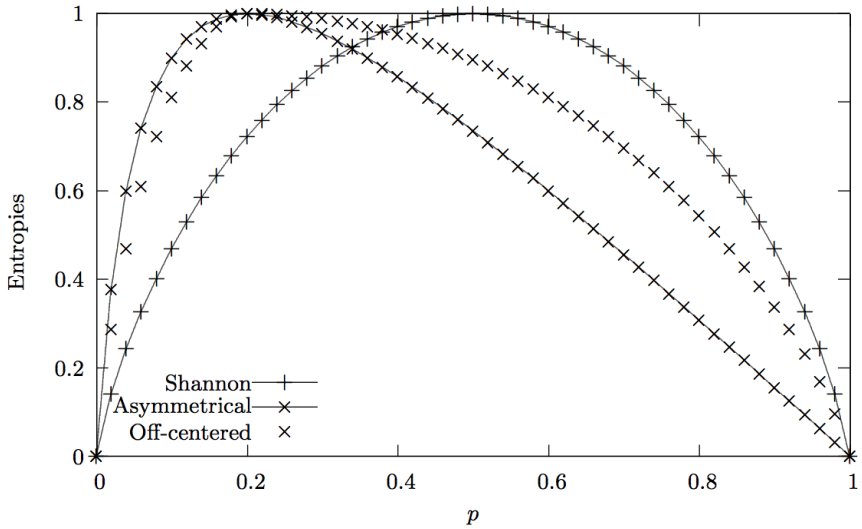

FIGURE 3.5 – Comparaison d'entropies [LLV07]

La Figure 3.5 montre les valeurs des différentes entropies en fonction de

10. c'est-à-dire la valeur 1
11. Pour ce exemple et afin de faciliter la lecture, nous posons $p = d(C_t, T)$.

l'évolution de la distribution d'un ensemble d'instances. Si la valeur du θ est de 0.2 et que nous sommes face à un ensemble d'instances de distribution $p = 0.2$, l'entropie décentrée sera maximale en ce point. L'entropie de Shannon est par contre maximale en $p = 0.5$, étant donné le caractère non-paramétrable de celle-ci.

Imaginons que nous devions, lors du processus de segmentation quantitatif de l'arbre de décision sur un jeu de données de distribution [12] (10,30), choisir entre deux sous-ensembles d'instances, A et B, ayant respectivement des distributions $(4, 4)$ et $(3, 5)$.
Comparons les choix respectifs de l'entropie de Shannon (voir Formule 2.7) et de l'entropie décentrée (voir Formule 3.9). L'entropie de Shannon donne :

$$\begin{aligned} Ent(A) &= -\sum_{t=1}^{q} d(C_t, A) \log_2(d(C_t, A)) \\ &= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} \\ &= 1 \end{aligned}$$

$$\begin{aligned} Ent(B) &= -\sum_{t=1}^{q} d(C_t, B) \log_2(d(C_t, B)) \\ &= -\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8} \\ &= 0.954 \end{aligned}$$

Calculons à présent l'entropie décentrée des sous-ensembles A et B. Nous paramétrons θ en lui donnant la valeur de distribution de la classe minoritaire du jeu de données initial, c'est-à-dire $\theta = \frac{1}{4}$.

Pour le sous-ensemble A :

Nous devons commencer par calculer le π en sachant que $p = \frac{1}{2}$ et est donc compris entre $\theta \leq p \leq 1$,

12. (classe positive ; classe négative)

$$\begin{aligned} \pi &= \frac{p+1-2\theta}{2(1-\theta)} \\ &= \frac{\frac{1}{2}+1-(2\frac{1}{4})}{2(1-\frac{1}{4})} \\ &= \frac{2}{3} \end{aligned} \tag{3.10}$$

A partir du π, nous pouvons calculer $H_\theta(p)$:

$$\begin{aligned} H_\theta(\frac{1}{2}) &= -\pi \log_2 \pi - (1-\pi)\log_2(1-\pi) \\ &= -\frac{2}{3}\log_2 \frac{2}{3} - (\frac{1}{3})\log_2(\frac{1}{3}) \\ &= 0.918 \end{aligned} \tag{3.11}$$

Pour le sous-ensemble B :

Nous pouvons calculer le π en sachant que $p = \frac{3}{5}$ et est donc compris entre $\theta \leq p \leq 1$:

$$\begin{aligned} \pi &= \frac{p+1-2\theta}{2(1-\theta)} \\ &= \frac{\frac{3}{8}+1-(2\frac{1}{4})}{2(1-\frac{1}{4})} \\ &= \frac{7}{12} \end{aligned} \tag{3.12}$$

A partir du π, nous pouvons calculer $H_\theta(p)$:

$$\begin{aligned} H_\theta(\frac{3}{5}) &= -\pi \log_2 \pi - (1-\pi)\log_2(1-\pi) \\ &= -\frac{7}{12}\log_2 \frac{7}{12} - (\frac{5}{12})\log_2(\frac{5}{12}) \\ &= 0.98 \end{aligned} \tag{3.13}$$

A partir de ces calculs, nous pouvons voir que l'entropie de Shannon préférera choisir le sous-ensemble B au sous-ensemble A tandis que l'entropie décentrée préférera l'ensemble A à l'ensemble B.

3.6.2 Discrétisation et entropie décentrée

Afin de pouvoir discrétiser au moyen de l'entropie décentrée, nous avons modifié l'algorithme de [FI93] permettant de discrétiser en utilisant l'entropie de Shannon ainsi que le principe de Description de Longueur Minimale.

Le fonctionnement initial de cet algorithme de discrétisation au moyen de l'entropie, lorsqu'il discrétise de façon binaire est le suivant :
Soit la matrice S_M (Section 2.1) composée de m instances et n attributs et ayant $n-1$ attributs indépendants quantitatifs (l'attribut de la colonne n étant l'attribut cible) et supposons que nous devions discrétiser l'attribut 1, les étapes à effectuer pour une discrétisation binaire sont les suivantes :

1. Nous trions de manière croissante l'ensemble des valeurs $(a_{1,1}, a_{2,1} \ldots, a_{m,1})$.
2. Nous calculons les cut points de telle sorte que, j étant l'ordre de l'instance après le tri, on a, pour tout cut point c : $c = \frac{a_{j,1} a_{j+1,1}}{2}$.
3. Nous calculons l'entropie de chaque partition (une partition par boundary point).
4. Nous sélectionnons le boundary point b_b (la "meilleure" partition) ayant la meilleure valeur d'ACE.
5. Si la condition du principe de Description de Longueur Minimale est vraie alors nous discrétisons suivant le boundary point b_b.

Nous pouvons **généraliser** aisément à partir de ce processus de discrétisation binaire en appliquant récursivement celui-ci sur chaque intervalle de chaque partition créée.

La Figure 3.6 illustre la version n-aire du processus présenté ci-dessus. Dans celui-ci sont présentes des instances triées selon un attribut. Cinq boundary points $\{b_1, b_2, b_3, b_4, b_5\}$ sont identifiés. La distribution des sous-ensembles bornés par les boundary points est la suivante [13] :

- $\leftarrow; b_1]$: distribution (a ;,b)
- $]b_1; b_2]$: distribution (c,d)
- $]b_2; b_3]$: distribution (e,f)
- $]b_3; b_4]$: distribution (g,h)
- $]b_4; b_5]$: distribution (i,j)
- $]b_5; \rightarrow$: distribution (k,l)

13. de forme : (classe positive, classe négative)

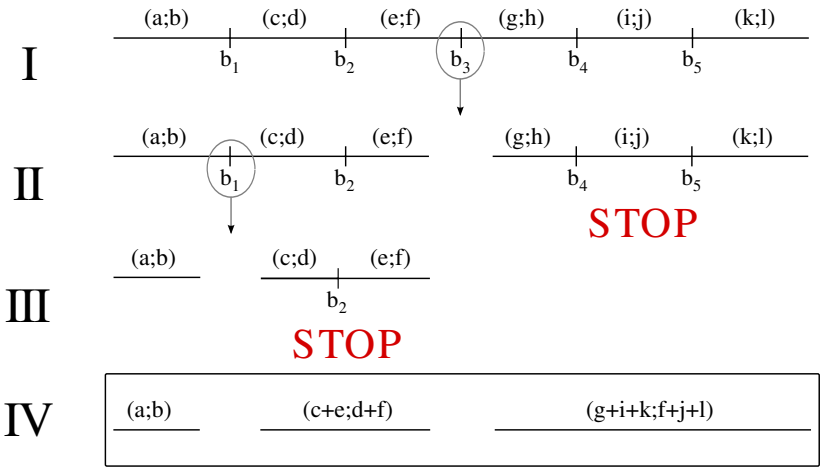

FIGURE 3.6 – Discrétisation n-aire en utilisant la méthode de discrétisation de [FI93]

Lors de l'étape **I**, nous sélectionnons le boundary point b_3 car il possède la meilleure valeur d'ACE. Ce boundary point passe le test MDL et l'intervalle est divisé en deux parties.
Lors de l'étape **II**, le processus de discrétisation binaire est appliqué sur deux intervalles : celui comprenant les boundary points b_1 et b_2 et celui composé des boundary points b_4 et b_5. Dans l'intervalle de b_1 b_2, le boundary point b_1 passe avec succès le test MDL, contrairement aux boundary points b_4 et b_5. Celui-ci est donc divisé en deux parties.
Durant l'étape **III**, le boundary point b_2 ne passe pas le test MDL, nous ne discrétisons par conséquent pas.
L'étape **IV** montre la partition finale, composée de trois sous-ensembles créés grâce aux boundary points b_1 et b_3. La partition ainsi créée contient trois sous-ensembles :

- $\leftarrow; b_1]$: distribution (a ;b)
- $]b_1; b_3]$: distribution (c+e ;d+f)
- $]b_3; \rightarrow]$: distribution (g+i+k ; f+j+l)

Le processus de discrétisation utilisant l'entropie et le MDL peut être qualifié de **supervisé**, **hiérarchique**, **top-down** et **non paramétrique**.

Nous avons adapté ce processus de discrétisation afin qu'il prenne en compte l'entropie décentrée. Pour cela, nous avons supprimé le principe de Description

de Longueur Minimale, qui, comme énoncé précédemment, ne donnait pas de résultats satisfaisants lorsqu'il était combiné à la discrétisation locale. Nous avons ajouté un critère de type arité fixe afin de remplacer celui-ci et nous avons restreint le processus à une discrétisation uniquement binaire.

3.6.3 Arbre et entropie décentrée

Cette deuxième instance de l'algorithme de base de création d'arbres de décision intègre la mesure entropie décentrée [LLV07] à l'algorithme de discrétisation utilisant l'entropie et le e principe de Description de Longueur Minimale [FI93]. Nous avons choisi de supprimer de ce processus de discrétisation le principe de Description de Longueur Minimale et nous l'avons remplacé par l'arité fixe. L'arité de l'arbre n'est par conséquent pas déterminée de manière "dynamique" comme lors de l'utilisation d'une méthode de discrétisation associée au principe initial.

Etant donnée l'utilisation de l'entropie décentrée, l'arbre de décision à créer doit avoir pour paramètre un θ fixant la valeur rendant l'entropie décentrée maximale. Dans le cadre de ce travail, ce paramètre est fixé au début du processus de construction de l'arbre de décision et ne change pas. Nous pouvons imaginer des améliorations fixant un θ particulier pour la segmentation de chaque noeud ou encore une méthode permettant de choisir le meilleur θ pour chaque segmentation.

3.6.4 Choix

Nous avons décidé d'intégrer cette mesure dans le processus de segmentation de notre second arbre de décision car plusieurs expérimentations ont montré qu'elle obtenait de bons résultats lors de la création d'arbres de décision (ou le processus de segmentation est qualitatif) à partir de jeux de données asymétriques [Mar, TNPLL08].

Nous avons donc ajouté cette mesure à une méthode de discrétisation afin de faire plusieurs expérimentations. Cette mesure, du fait de l'introduction du paramètre θ rend l'arbre de décision plus paramétrable et permet des expérimentations plus approfondies.

4 Résultats

Dans cette partie, nous présentons les résultats obtenus par les différentes instances de l'algorithme développé.
Nous commençons par exposer les jeux de données et la méthode de test utilisés avant d'expliquer en détails les différentes expérimentations effectuées. Les instances développées sont testées au moyen de plusieurs jeux de données et leurs performances sont comparées d'une part avec l'algorithme C4.5, algorithme implémenté dans Weka [1] et d'autre part avec l'algorithme CCP-C4.5, algorithme basé sur C4.5, spécialement conçu pour les jeux de données asymétriques (et détaillé dans [LCCC10]).

4.1 Machines de test

La machine utilisée pour l'entièreté des tests est un Macbook Pro disposant d'un processeur i5, de 4GB de mémoire Ram et tournant sous Mac OSX Leopard.
Un problème de compatibilité de librairies entre le programme Splitter [2] [Rou01] et OSX Leopard nous a contraints à exécuter notre logiciel sur une distribution Ubuntu [3] virtualisée.

1. nommé *j48*, dernière version non-commerciale de cet algorithme de création d'arbres de décision
2. programme inséré dans notre logiciel
3. version 10.10

4.2 Jeux de donnés utilisés

Les jeux de données utilisés sont issus d'un répertoire de jeux de données fournis par la NASA [rep].
Ces jeux de données appartiennent au domaine de l'ingénierie logicielle et sont utilisés afin de prédire une défaillance d'un module d'un logiciel en fonction des différents paramètres (métriques) de celui-ci, telle que sa taille, sa complexité,...

Chaque instance de ce jeu de données est un module d'un logiciel particulier qui contient un attribut cible "Defect", attribut indiquant si le module a déjà oui ou non montré des signes de défectuosité (il a des défauts ou il est exempt de défauts). Le but est de construire un classifieur à partir de ces jeux de données afin de pouvoir prédire si, selon ses caractéristiques, un module présente une défectuosité ou non. Les quatre jeux de données utilisés pour les expérimentations ont tous la même structure. Ils contiennent 22 attributs indépendants, attributs qui mesurent la taille du module ou encore sa complexité. La valeur d'attribut cible "Defect=oui" est la classe minoritaire de ces jeux de données.

Chaque jeu de données contient les modules d'un logiciel particulier :

- **cm1** : logiciel intégré dans une navette spatiale et écrit en C.
- **kc1** : système écrit en C++ et implémentant le management de stockage afin de recevoir et d'envoyer les données collectées durant le vol.
- **kc2** : similaire à kc1 (autre partie du projet).
- **pc1** : logiciel de vol d'un satellite orbitant autour de la Terre.

Les jeux de données **cm1**, **kc1**, **kc2** correspondent à un manque relatif de données (Définition 2.7.5) alors que **pc1** correspond à un manque absolu de données (Définition 2.7.4). Des informations supplémentaires à propos de ces jeux de données sont disponibles dans la Table 4.1.

Jeu de données	Instances	Attributs	Classe minoritaire
`cm1`	498	22	9,8 %
`kc1`	2109	22	15,5%
`kc2`	522	22	20,5%
`pc1`	1109	22	6,9%

TABLE 4.1 – Jeux de données utilisés pour l'expérimentation

4.3 Validation croisée

Afin d'évaluer les performances des classifieurs développés, nous avons utilisé la **validation croisée**. Nous avons expliqué à plusieurs reprises le processus de base de la validation croisée, c'est-à-dire la construction d'un modèle (d'un classifieur) à partir d'un jeu de données d'apprentissage et l'application de celui-ci à un jeu de données de test afin d'évaluer ses performances. L'inconvénient de cette technique est que l'évaluation du modèle dépend fortement de la division du jeu de données en jeux de données d'apprentissage et de test.

Une méthode permettant d'être moins dépendant de la division du jeu de données est la **k-validation croisée** [RTL09]. Celle-ci divise le jeu de données initial en k morceaux et produit k classifieurs.

Le processus de k-validation croisée se déroule comme suit :

1. Segmentation du jeu de données en k-morceaux qui ont une distribution similaire à la distribution du jeu de données initial (dans le cas d'une distribution similaire, la validation croisée est dite *stratifiée*)
2. Pour chacune des k itérations, nous créons le classifieur sur les $k-1$ sous-ensembles (le reste) et l'appliquons au morceau k.

La Figure 4.1 permet d'illustrer un exemple de k-validation croisée. Dans celle-ci, le jeu de données est divisé en trois sous-ensembles. Sur chaque sous-ensemble est appliqué le modèle créé à partir des deux autres parties. Nous pouvons voir que cette technique permet de prédire la classe de chaque instance du jeu de données une fois.

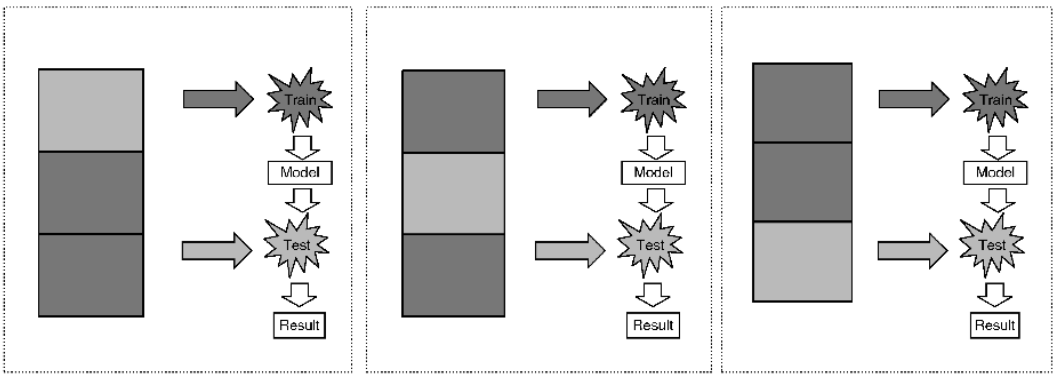

FIGURE 4.1 – Exemple de 3-validation croisée [RTL09]

Dans le cadre de ce travail, nous avons bien entendu utilisé la k-validation

croisée stratifiée afin d'avoir des jeux de données représentatifs et comprenant le même pourcentage d'instances de classe minoritaire.

Tous les algorithmes expérimentés dans le cadre de ce travail ont été testés au moyen de la méthode 5X2 validation croisée, . Celle-ci exécute cinq fois une 2-validation croisée et fait la moyenne des résultats obtenus. Afin de garder la même proportion d'instances de classe minoritaire dans les morceaux créés par la k-validation croisée que dans le jeu de données initial, nous avons bien entendu utilisé la version stratifiée de la k-validation croisée.
Par exemple, lorsqu'il s'agit de calculer l'aire sous la courbe de la 5-validation croisée, nous exécutons cinq fois la 2-validation croisée, ce qui crée cinq courbes ROC similaires à celle de la Figure 4.2. Ensuite, nous calculons pour chaque courbe ROC l'aire sous la courbe. Enfin, nous calculons la moyenne des cinq aires sous la courbe.

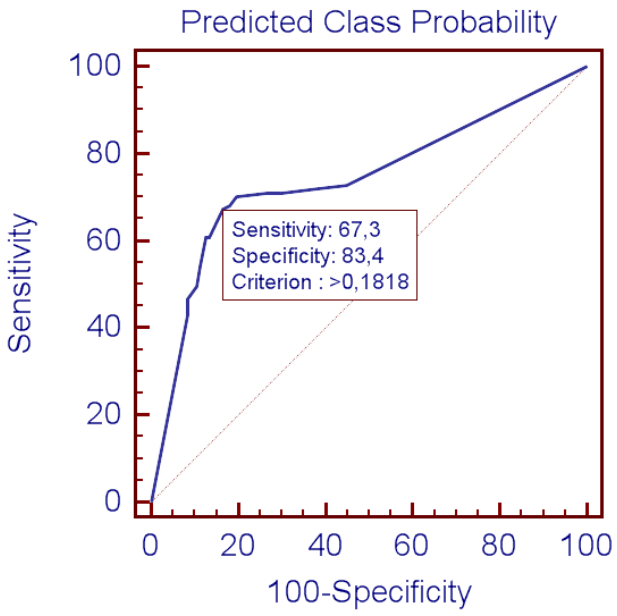

FIGURE 4.2 – Exemple de courbe ROC obtenue au moyen de [SZDC95]

4.4 Résultats des expérimentations effectuées

Nous avons choisi d'effectuer notre évaluation des performances des différentes instances de l'algorithme de base en deux temps. Dans un premier temps, nous exécutons les deux instances de l'algorithme (en utilisant différents paramètres) et les algorithmes externes (C4.5, CCP-C4.5) sur des jeux de données

afin de créer des arbres de décision non élagués. Dans un second temps, nous exécutons les deux instances de l'algorithme sur des jeux de données afin de créer des arbres élagués. Cette scission des expérimentations en deux parties est motivée par plusieurs raisons :

- Pouvoir évaluer les performances de la segmentation quantitative en se focalisant sur la fonction d'impureté spécialement dédiée à l'asymétrie : l'entropie décentrée.
- Comparer les différentes méthodes sans élagage et avec élagage
- Comparer les deux méthodes d'élagage, le post-élagage au moyen du test exact de Fisher et le "Error Based Pruning" afin de confirmer la suprématie de la première méthode et ainsi confirmer les conclusions obtenues par [LCCC10].

Pour chacune des deux parties, nous analysons les classifieurs produits au moyen des critères suivants :

- **AUC** (Section 2.5.1) : Nous calculons l'aire sous la courbe ROC. Dans le cas de l'utilisation d'une 5X2 validation croisée, nous calculons l'aire sous la courbe de chaque 2-validation croisée avant de faire la moyenne pour obtenir l'aire sour la courbe du classifieur pour un jeu de données particulier.
- **nombre** de **feuilles** (règles) de l'arbre de décision..

Les techniques comparées sont abréviées de la façon suivante :

- **ATSE2** : Instance de l'algorithme de base utilisant le Training set Error (Section 3.5) et une arité bornée de 2.
- **ATSE3** : Instance de l'algorithme de base utilisant le Training set Error (Section 3.5) et une arité bornée de 3.
- **AENT** : Instance de l'algorithme de base utilisant l'entropie de Shannon.
- **AOCE** : Instance de l'algorithme de base utilisant l'entropie décentrée (Section 3.6).
- **C4.5** : Algorithme de création d'arbres de décision implémenté dans Weka [Qui93].
- **C4.5-CCP** : Algorithme de création d'arbres de décision créé par [LCCC10].

Les paramètres utilisés pour les analyses des instances de l'algorithme conçu dans la cadre de ce travail, à savoir les (sous-)instances suivantes : **ATSE**, **AENT2**, **AENT3** et **AOCE** sont :

- threshold de 2.
- arité de 2 (**ATSE**, **AENT**, **AOCE**) ou 3 (**ATSE**).
- pas d'utilisation de la correction de Laplace.

Les paramètres utilisés pour les algorithmes développés par des tiers, c'est-à-dire **C4.5**, **C4.5-CCP** sont :

- threshold de 2
- pas d'utilisation de la correction de Laplace

4.4.1 Sans élagage

Dans cette partie, nous testons et évaluons des algorithmes de création d'arbres de décision sans élaguer ceux-ci. Nous comparons les arbres de décision produits par les différentes méthodes (algorithmes) selon leur valeur d'AUC et selon le nombre de feuilles que ceux-ci produisent.

4.4.1.1 Aire sous la courbe ROC

La Table 4.2 donne la valeur d'AUC pour chaque application d'un classifieur donné à un des quatre jeux de données. Nous avons choisi de calculer l'AUC de chaque classifieur afin d'avoir une vue globale des performances de celui-ci. Notre comparaison entre les différents algorithmes suivant l'AUC se fait en deux temps. Nous commençons d'abord pas commenter la Table 4.2 de manière intuitive [4] avant d'appliquer le test de Friedman afin de pouvoir faire des comparaisons multiples [5].

Jeu de données	ATSE2	ATSE3	AENT	AOCE	C4.5	C4.5-CCP
cm1	0.545 (6)	0.547 (5)	0.581 (2)	0.559 (4)	0.579 (3)	0.644 (1)
kc1	0.550 (5)	0.520 (6)	0.616 (2)	0.604 (3)	0.599 (4)	0.659 (1)
kc2	0.659 (3)	0.586 (6)	0.660 (2)	0.652 (4)	0.617 (5)	0.691 (1)
pc1	0.551 (5)	0.546 (6)	0.658 (2)	0.626 (4)	0.643 (3)	0.717 (1)

TABLE 4.2 – Valeur d'AUC de chaque jeu de données suivant un algorithme particulier

A partir de la Table 4.2, nous faisons les constatations (intuives) suivantes. La méthode donnant les moins bons résultats pour une majorité de jeux de données est **ATSE3**. Cela peut être expliqué par le fait que discrétiser suivant une arité de trois produit un modèle très surajusté et qui est fragmenté de façon excessive. Notre but, lors de l'utilisation du Training Set Error et d'une arité de trois était surtout de voir les performances de celui-ci après l'application d'une méthode de post-élagage. Les performances d'**ATSE3** sont globalement assez faibles mais il serait intéressant d'étudier plus finement les règles produites à partir de ce classifieur afin de trouver des règles intéressantes pour l'apprentissage

ATSE3 obtient de moins bonnes performances que **ATSE2** sur trois des quatre

4. Nous assignons à cet égard un "classement" à chaque aire sous la courbe afin de faciliter la comparaison.

5. Etant donné le nombre limité de jeux de données sur lesquels nous faisons les expérimentations, la puissance du test de Friedman est limitée et ses résultats sont à prendre précautionneusement.

jeux de données.

Comparons à présent les deux méthodes utilisant deux formes d'entropies différentes : l'entropie de Shannon et l'entropie décentrée. Nous nous attendions à ce que l'entropie décentrée donne de meilleures performances vu le caractère asymétrique des quatre jeux de données expérimentés mais c'est pourtant **AENT** qui semble donner les meilleurs résultats.
Lorsque nous comparons les classifeurs conçus dans le cadre de ce travail avec les deux classifieurs externes, à savoir **C4.5** et **C4.5-CCP**, nous pouvons voire qu'**AENT** et **AOCE** offrent des performances similaires voir légèrement supérieures (en tout cas pour **AENT**) que l'algorithme le plus populaire, **C4.5**. La méthode ayant les meilleurs résultats sur les quatre jeux de données est **C4.5-CCP**, méthode testée également dans [LCCC10] et possédant également dans cet article les meilleures performances parmi l'ensemble des méthodes expérimentées.

	ATSE2	**ATSE3**	**AENT**	**AOCE**	**C4.5**	**C4.5-CCP**
ATSE2	X	-	-	<	-	<
ATSE3	-	X	<	<	<	<
AENT	>	>	X	>	>	-
AOCE	-	>	<	X	-	<
C4.5	-	>	<	-	X	<
C4.5-CCP	>	>	-	>	>	X

TABLE 4.3 – Comparaisons multiples au moyen du test de Friedman

Le test de Friedman permet de savoir s'il y a une différence entre les différentes variables. Si la p-value est plus petite que 0.05, l'hypothèse nulle (pas de différence entre les variables) est rejetée.

Nous avons utilisé les notations :

- **"-"** : lorsque le test de Friedman arrive à la conclusion qu'il n'y a pas de différences entre les variables.
- **">"** : lorsque le test de Friedman arrive à la conclusion qu'il y a une différence entre les deux variables (nous mettons le signe ">" pour énoncer le fait que le classifieur "ligne" donne de meilleurs résultats que le classifieur "colonne").
- **"<"** : lorsque le test de Friedman arrive à la conclusion qu'il y a une différence entre les deux variables (nous mettons le signe "< "pour énoncer le fait que le classifieur "ligne" donne de moins bons résultats que le

classifieur “colonne”).

D’après la Table 4.3, le test de Friedman confirme ce que nous pensions intuitivement : **ATSE3** est l’ (la sous) instance qui donne les moins bonnes valeurs d’AUC.
Les deux classifieurs donnant les meilleures performances sont **ATSE** et **C4.5-CCP**. Le test de Friedman n’arrive par à rejeter, pour ces deux-là, l’hypothèse H_0.
Les autres classifieurs n’étant pas départagés par le test de Friedman sont **ATSE2** et **ATSE3**, **ATSE2** et **AENT**, **ATSE2** et **C4.5**, **C4.5** et **AOCE**.
La méthode développée dans le cadre de ce travail qui semble donner les meilleurs résultats en terme d’AUC pour ce critère est **AENT**.

4.4.1.2 Nombre de règles

Nous comparons à présent le nombre de règles produites par chaque classifieur pour chaque jeu de données. Le nombre de règles produites permet de se faire une idée de la complexité du modèle ainsi que de la facilité avec laquelle nous pouvons interpréter celui-ci. Nous pouvons visualiser le nombre de règles produites par chaque classifieur pour chaque jeu de données grâce à la Table 4.4.

Jeu de données	ATSE2	ATSE3	AENT	AOCE	C4.5	C4.5-CCP
cm1	6 (1)	48 (4)	50 (5)	55 (6)	36 (2)	26 (3)
kc1	16 (1)	227 (4)	240 (5)	286 (6)	167 (2)	205 (3)
kc2	10 (1)	69 (6)	66 (5)	65 (4)	38 (2)	47 (3)
pc1	5 (1)	76 (6)	69 (4)	75 (5)	64 (3)	32 (2)

TABLE 4.4 – Nombre de règles des arbres de décision produits pour chaque jeu de données - sans élagage

Le classifieur possédant sans conteste le moins de règles est **ATSE2**. Il est par conséquent le plus facile à appréhender et à interpréter. Le faible nombre de règles de ce classifieur peut s’expliquer par l’arité bornée combinée à une arité de deux.
A contrario, lorsque l’arité vaut trois, le classifieur **ATSE3** produit un très grand nombre de règles, du fait de son arité, qui, pour de nombreux noeuds, segmentent ceux-ci en trois fils. L’arbre grandit donc assez rapidement “en largeur”.
Cette arbre est généralement moins profond que les arbres produits par les clas-

sifieurs **AENT**, **AOCE**, **C4.5**, **C4.5-CCP**. Du fait de l'aspect large des arbres de décision produits avec **ATSE3**, nous pouvons dire que souvent les règles du classifieur **ATSE3** auront une partie antécédente de taille moins importante que les classifieurs utilisant la discrétisation binaire (excepté pour **ATSE2**), c'est-à-dire les classifieurs **AENT**, **AOCE**, **C4.5** et **C4.5-CCP**.
Les classifieurs possédant le plus de règles sont les classifieurs spécialement conçus pour les jeux de données asymétriques. Ce résultat est logique étant donné que ces classifeurs veulent éviter d'être trop généraux et ciblent de façon précise les instances minoritaires. Il peut résulter de ce type de règle la production de "Small Disjuncts" (Définition 2.7.3). Pour ces classifieurs, le post-élagage est primordial et va permettre de supprimer une majorité de règles statistiquement non-significatives.

4.4.2 Avec élagage

Dans cette partie, nous testons et évaluons les algorithmes de création d'arbres de décision et nous utilisons une méthode d'élagage [6] afin de réduire la taille des arbres produits.

De façon identique à la Section 4.4.1, nous comparons les arbres de décision produits par les différentes méthodes (algorithmes) selon leur valeur d'AUC et suivant le nombre de feuilles que ceux-ci produisent.

4.4.2.1 Aire sous la courbe

Nous donnons les valeurs d'AUC des arbres de décision créés par des classifieurs pour les jeux de données grâce à la Table 4.5.

Jeu de données	ATSE2	ATSE3	AENT	AOCE	C4.5	CCP-C4.5
cm1	0.545 (4)	0.539 (5)	0.579 (2)	0.568 (3)	0.58 (6)	0.634 (1)
kc1	0.564 (5)	0.556 (6)	0.673 (3)	0.630 (4)	0.633 (2)	0.675 (1)
kc2	0.677 (3)	0.627 (6)	0.726 (2)	0.664 (5)	0.670 (4)	0.691 (1)
pc1	0.551 (6)	0.561 (5)	0.726 (1)	0.628 (4)	0.647 (3)	0.725 (2)

TABLE 4.5 – Valeur d'AUC lors de l'élagage des arbres de décision

	ATSE2	ATSE3	AENT	AOCE	C4.5	C4.5-CCP
ATSE2	X	-	<	-	<	<
ATSE3	-	X	<	<	<	<
AENT	>	>	X	>	-	-
AOCE	-	>	<	X	-	<
C4.5	>	>	-	-	X	<
C4.5-CCP	>	>	-	>	>	X

TABLE 4.6 – Comparaisons multiples au moyen du test de Friedman

Nous ne distinguons par de bouleversements dans les performances des classifieurs suite à l'application d'une méthode d'élagage. Nous pouvons néanmois comparer **C4.5** utilisant la méthode d'élagage "error based pruning" et les autres algorithmes qui sont tous dotés de la méthode de post-élagage au moyen du test exact de Fisher.
Alors que les expérimentations effectuées dans [LCCC10] montraient l'élagage

6. soit le "error-based pruning" ou soit le post-élagage utilisant le test exact de Fisher

attaché à la méthode C4.5 comme moins performant, l'étude des deux techniques d'élagage nous montre que sur les quatre jeux de données, elles se comportent approximativement de la même manière. Par exemple, lorsque nous comparons l'aire sous la courbe pour **C4.5** avec et sans méthode d'élagage, nous constatons que l'aire sous la courbe de **C4.5** est supérieure pour les quatre jeux de données lorsqu'une méthode d'élagage esst utilisée.
Lorsque nous mettons en parallèle les performances de **C4.5-CCP** avec et sans l'utilisation d'une méthode d'élagage, nous constatons que **C4.5-CCP** a une aire sous la courbe supérieure pour trois jeux de données sur les quatre lorsqu'il contient une méthode d'élagage. D'après nos expérimentations sur les quatre jeux de données les deux méthodes d'élagage ont des performances assez similaires.

4.4.2.2 Règles

La Table 4.7 présente le nombre de règles de tous les arbres de décision produits à partir des jeux de données.

Jeu de données	ATSE2	ATSE3	AENT	AOCE	C4.5	CCP-C4.5
cm1	5 (1)	21 (4)	33 (4)	43 (5)	14 (2)	17 (3)
kc1	13 (1)	110 (4)	144 (5)	203 (6)	96 (2)	109 (3)
kc2	6 (2)	46 (5)	39 (4)	51 (6)	37 (1)	30 (3)
pc1	5 (1)	26 (3)	48 (5)	75 §6)	10 (2)	17 (4)

TABLE 4.7 – Nombre de règles des arbres de décision produits pour chaque jeu de données - avec élagage

5

Conclusion et perspectives

5.1 Conclusion

L'objectif principal de ce mémoire était de concevoir et d'implémenter un classifieur binaire entièrement dédié à un type particulier de jeux de données. Ce type de jeux de données contient deux spécificités. Premièrement, une classe de celui-ci est largement sous-représentée par rapport à l'autre classe, ce genre de jeux de données est dit asymétrique. Deuxièmement, les attributs indépendants de ces jeux de données sont quantitatifs.

Afin d'atteindre cet objectif, le classifieur que nous avons conçu et implémenté dans ce travail est un algorithme de création d'arbres de décision possédant plusieurs instances, chacune ayant ses spécificités et un objectif particulier. Les deux instances développées ont en commun le fait d'utiliser une méthode de post-élagage utilisant le test exact de Fisher.

La première instance de l'algorithme de base utilise la mesure d'impureté Training Set Error. Celle-ci a été utilisée afin de pouvoir être comparée avec la deuxième instance et ainsi de mesurer les gains potentiels de cette dernière. Nous l'avons également utilisée dans le but de pouvoir créer des arbres de décision d'arité supérieure à deux. Une raison supplémentaire de son utilisation est le fait que nous ayons la possibilité de faire évoluer celle-ci dans le futur

vers une discrétisation sensible au coûts [BV98] et d'intégrer celle-ci dans notre solution.

La deuxième instance utilise une mesure plus adaptée à la classe minoritaire : l'entropie décentrée. C'est cette instance qui peut être considérée comme l'instance entièrement dédiée aux jeux de données étudiés dans ce travail.

L'ensemble des technique intégrées dans notre solution ont été choisies car elles offrent de bonnes performances par rapport à la classe minoritaire. Nous voulions tester les répercussions qu'ont les attributs indépendants quantitatifs sur les jeux de données asymétriques. C'est le cas pour l'entropie décentrée [LLV07, Mar] ou encore le test exact de Fisher [LCCC10].

Après une explication détaillée de tous les concepts permettant de comprendre en profondeur la solution développé ainsi qu'un chapitre présentant la solution à proprement parler, nous avons expérimenté notre solution (Chapitre 4). Cette expérimentation s'est faite en deux temps. Premièrement, nous avons voulu comparer les résultats des différents algorithmes (en comparant nos instances avec deux algorithmes externes, c'est-à-dire non développés dans ce mémoire) sans élaguer les arbres de décision produits afin de voir si nos instances, spécialement celle utilisant l'entropie décentrée, augmentait les performances de manière notable.
Ensuite, nous avons comparé les résultats des différents algorithmes en élaguant les arbres produits.

A partir de ces expérimentations, nous pouvons faire plusieurs constatations. Premièrement, pour les jeux de données testés, nous ne pouvons affirmer que la deuxième instance de notre solution (utilisant l'entropie décentrée) surpasse une variante de celle-ci utilisant l'entropie de Shannon.
Deuxièmement, alors que l'article [LCCC10], après avoir testé les méthodes d'élagage de C4.5 et du post-élagage utilisant le test exact de Fisher affirmait que la deuxième méthode surpassait la première, nous ne sommes pas en mesure, de par les expérimentations effectuées dans ce travail, d'être aussi catégoriques.
Troisièmement, la deuxième instance de notre solution utilisant l'entropie décentrée, bien que possédant d'honorables résultats lors des différentes expérimentations et étant en moyenne meilleure que C4.5 sur les critères testés, cette instance n'arrive pas à égaler les performances de C4.5-CCP [LCCC10].

5.2 Perspectives

Ce mémoire constitue une base de travail intéressante pour toute personne désirant apprendre et appliquer plusieurs algorithmes de création d'arbres de décision à des fins de classification. Les directions que peut prendre ce travail dans le futur sont multiples. Quelques nouvelles perspectives sont présentées ci-dessous.

Nous divisons les perspectives en deux parties : les perspectives pouvant faire évoluer la solution de manière purement théorique et les perspectives d'évolution de notre logiciel.

Au niveau Theorique :

- Un arbre de décision est une structure qui peut être instable. Nous pourrions envisager de modifier notre solution afin que celle-ci crée des structures plus complexes telles que le bagging [Bre96] afin de combiner entre eux les arbres de décision. Cette combinaison pourrait réduire la variance et ainsi améliorer les prédictions.
- Nous pourrions également étendre nos algorithmes afin qu'ils soient en mesure d'appréhender des jeux de données dont l'attribut cible possède plus de deux valeurs différentes.
- Une autre piste d'évolution serait d'approfondir l'étude de l'entropie décentrée en construisant une méthode permettant de déterminer dynamiquement le paramètre θ, paramètre rendant l'entropie décentrée maximale. Nous pourrions par exemple développer cette méthode afin de l'appliquer à un algorithme de création d'arbres de décision et ainsi adapter le θ en fonction de la distribution de chaque noeud.
- Nous pourrions également comparer notre solution avec des méthodes diverses et variées telles que des algorithmes de Subgroup discovery, des algorithmes de sampling ou encore des algorithmes génétiques.
- Nous pensons que les arbres produits au moyen de l'entropie décentrée et de l'entropie ont une taille trop importante et que post-élagage au moyen du test exact de Fisher n'est pas en mesure de supprimer assez de noeuds. Une piste d'évolution future serait alors la conception d'un critère d'arrêt pour la discrétisation, tel que le Principe de Description de Longueur Minimale mais moins rescrictif que celui-ci afin de pouvoir être inséré dans un processus de discrétisation locale.
- Nous avons choisi la mesure Training Set Error car celle-ci peut évoluer rapidement vers une technique de discrétisation sensible aux coûts. Nous

pourrions concevoir un tel type de discrétisation.
– Nous pourrions tester les techniques de ce travail sur un plus grand nombre de jeux de données afin d'avoir des statistiques plus précises sur leurs performances.

Au niveau IMPLEMENTATION :
– Le logiciel a été conçu dans le but d'être réutilisé par des chercheurs d'un laboratoire. Ceux-ci peuvent donc le faire évoluer, ajouter de nouvelles techniques et méthodes à celui-ci,. . . ,

5.3 Apports

Nous listons à présent l'ensemble des **apports personnels**. Ces apports sont divisés en deux catégories : les apports purement théoriques et les apports relatifs à l'implémentation de la solution :

Au niveau THEORIQUE :

– Transformation de la méthode de discrétisation de [FI93] afin d'intégrer l'entropie décentrée.
– Conception d'un algorithme de création d'arbres de décision permettant de discrétiser localement selon les mesures suivantes : le *Training Set Error* et l'*entropie décentrée* et d'élaguer l'arbre produit au moyen du test exact de Fisher.
– Comparaison de notre solution avec celle venant de [LCCC10] et étant adaptée aux jeux de données asymétriques ainsi qu'avec C4.5.
– Brève comparaison des deux méthodes d'élagage.

Au niveau IMPLEMENTATION [1] :

– Création d'un logiciel afin de tester les différents algorithmes conçus et développés dans cet ouvrage et afin d'être utilisé par la laboratoire du lieu de stage. Ce logiciel est donc une base de travail sur laquelle vont venir se greffer de nouvelles solutions.

1. Annexe A.1

A

Annexe

A.1 Outil implémenté

Dans cette partie, nous décrivons le logiciel que nous avons conçu et implémenté. Celui-ci contient un algorithme de construction d'arbres de décision aisément adaptable qui a notamment servi à la création des arbres présentés tout au long du Chapitre 3. A titre de comparaison, cet algorithme est similaire à un algorithme tel que C4.5 (Section 2.3.7).

Nous avons choisi d'implémenter notre propre algorithme et un logiciel entièrement dédié aux arbres de décision dans le but de pouvoir avoir une plus grande maîtrise de celui-ci, une meilleure vue d'ensemble et ainsi d'augmentation ses capacités de modification et de réutilisation. Le logiciel implémenté permet donc de changer facilement de fonction d'impureté, de méthode d'élagage ou encore d'ajouter facilement un nouveau type de visualisation d'arbres de décision.

Ce logiciel intègre également deux (parties) de programme tiers, ce qui donne à notre solution un accès à toute une série de techniques supplémentaires et qui permettra dans le futur de faire des expérimentations plus poussées. Le programme construit permet également de visualiser les arbres de décision ainsi que les règles dérivant de ceux-ci.

Dans cette section, nous décrivons le fonctionnement de notre outil de manière chronologique en divisant celui-ci en trois étapes :

1. **Paramétrage** : choix des différents paramètres attachés à l'arbre de décision que nous voulons construire (Section A.1.1).
2. **Visualisation** : dans cette étape, nous pouvons visualiser ainsi qu'explorer l'arbre de décision produit ainsi que les règles dérivant de celui-ci (Section A.1.2).
3. **Application** : cette fonctionnalité du logiciel permet d'appliquer le modèle (l'arbre de décision) produit à un jeu de données de test (Section A.1.2.3).

A.1.1 Choix des paramètres

Afin de pouvoir faire de nombreuses expérimentations, notre logiciel permet de paramétrer l'algorithme de création d'arbres de décision.

Les paramètres pouvant être introduit par l'utilisateur sont les suivants :

- **dataset** : permet de sélectionner le jeu de données à partir duquel l'arbre de décision sera créé. Par défaut, l'arbre de décision sera construit et testé suivant un mécanisme de validation croisée. L'utilisateur peut aussi définir s'il veut que l'arbre de décision soit construit par ce jeu de données et testé par un jeu de données de test.

- **arité** : sélectionne l'arité de discrétisation utilisée pour la création de l'arbre de décision. Suivant la fonction d'impureté utilisée, l'arité peut être soit *fixée*, soit *bornée*.

- **threshold** : si le nombre d'instances du noeud courant est plus petit ou égal au seuil, celui-ci ne sera pas segmenté.

- **function** : la fonction d'impureté choisie pour la segmentation quantitative. Le logiciel permet de choisir l'une des trois fonctions d'impureté suivantes : le *Training Set Error*, *Entropie* et *Entropie décentrée*.

- **prune the tree** : l'utilisateur peut,grâce à ce paramètre choisir d'élaguer ou non l'arbre de décision produit.

– **epurateRules** : permet à l'utilisateur de spécifier s'il veut que les règles produites soient épurées selon la méthode énoncée dans la Section 3.4.3

La Figure A.1 montre l'interface graphique permettant d'introduire tous les paramètres. Celle-ci contient également une partie destinée à afficher l'ensemble des logs liés à la création de l'arbre de décision.

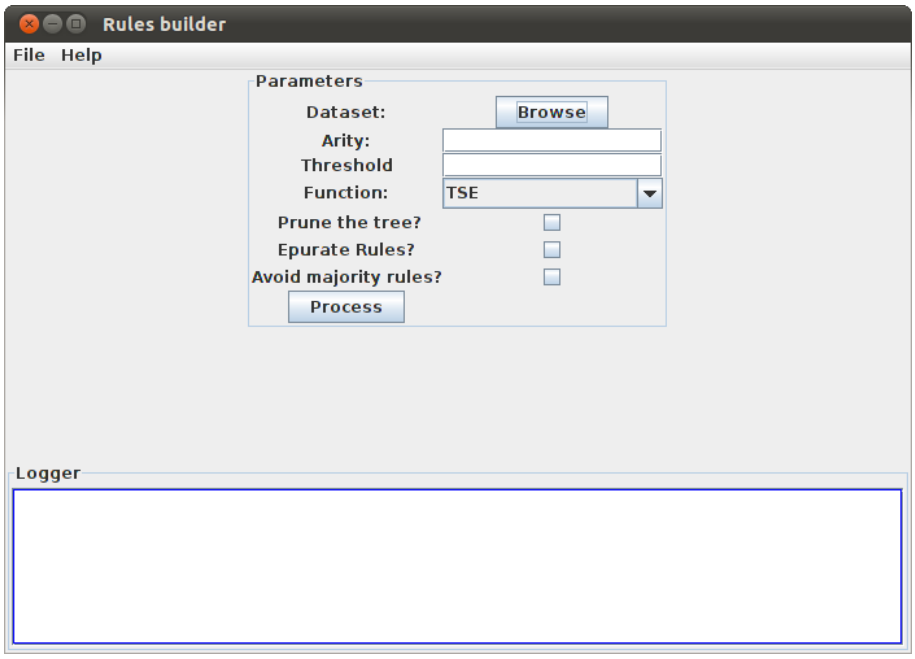

FIGURE A.1 – Fenêtre permettant de choisir les différents paramètres afin de construire l'arbre de décision

A.1.2 Visualisation et exploration

Après la construction de l'arbre de décision, nous avons à notre disposition toute une série d'informations sur celui-ci. Nous pouvons **visualiser** l'**arbre**, l'**explorer**, **parcourir** les **règles** dérivées de celui-ci ainsi que les **statistiques** générales de l'arbre de décision.

A.1.2.1 Arbre

L'interface graphique de visualisation et d'exploration d'abres de décision est illustrée par les Figure **??** et A.2. Chaque noeud de l'arbre possède un identifiant, affiche le nombre de fils ainsi que la distribution de l'ensemble d'instances attaché à celui-ci. Chaque noeud de l'arbre peut être exploré, c'est-à-dire que

nous pouvons voir les fils du noeud courant. La Figure A.2, nous montre un arbre de décision entièrement exploré.

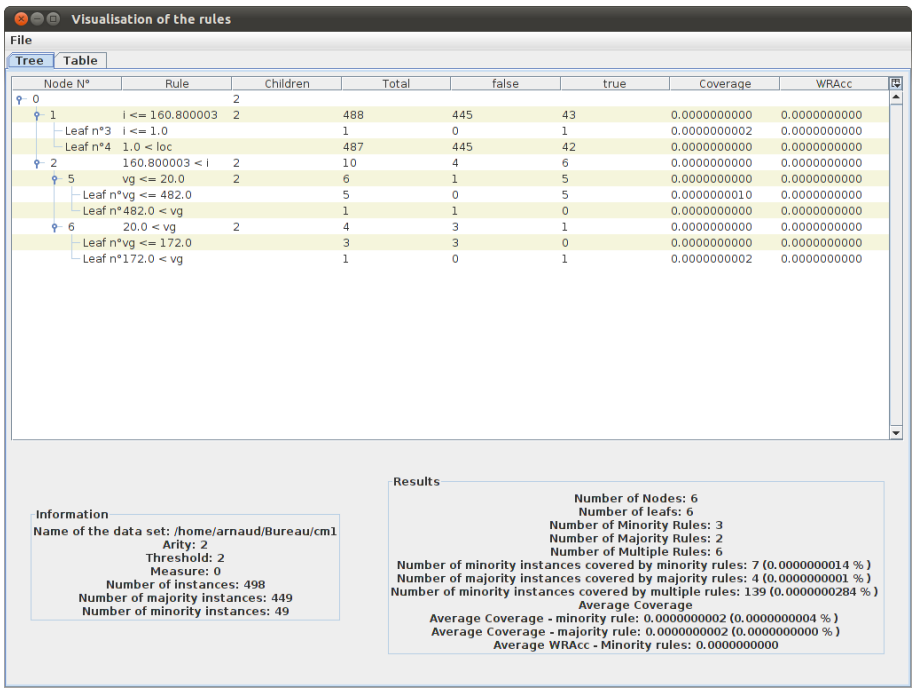

FIGURE A.2 – Fenêtre permettant d'explorer et de visualiser l'arbre de décision

Grâce à la Figure et A.2, nous pouvons voir que plusieurs informations sont disponibles à propos du jeu de données et de l'arbre de décision créé.
D'une part, certains renseignements tels que le nom du jeu de données, l'arité choisie, le threshold sélectionné, la mesure utilisée et la distribution du jeu de données inital sont présents.
D'autre part, plusieurs informations à propos de l'arbes sont disponibles telles que le nombre de noeuds (hors feuilles) de celui-ci, le nombre de feuilles ainsi que le résultat à plusieurs mesures quant à la qualité de l'arbre. Ces mesures portent sur le nombre de règles, les performances de classification.

Durant le processus de création de l'arbre de décision, un grand nombre de données sont enregistrées afin de permettre à l'utilisateur de comprendre plus en profondeur les mécanismes relatifs à sa création. Ces données permettent également à l'utilisateur, par exemple lors de l'analyse des instances présentes dans un noeud du jeu de données de trouver de nouvelles relations, de nouveaux principes par rapport aux données étudiées ("à la façon d'un système guidé par un expert"). La Figure A.3 donne une aperçu de toutes les données qui sont sauvegardées durant le processus de création de l'arbre de décision.

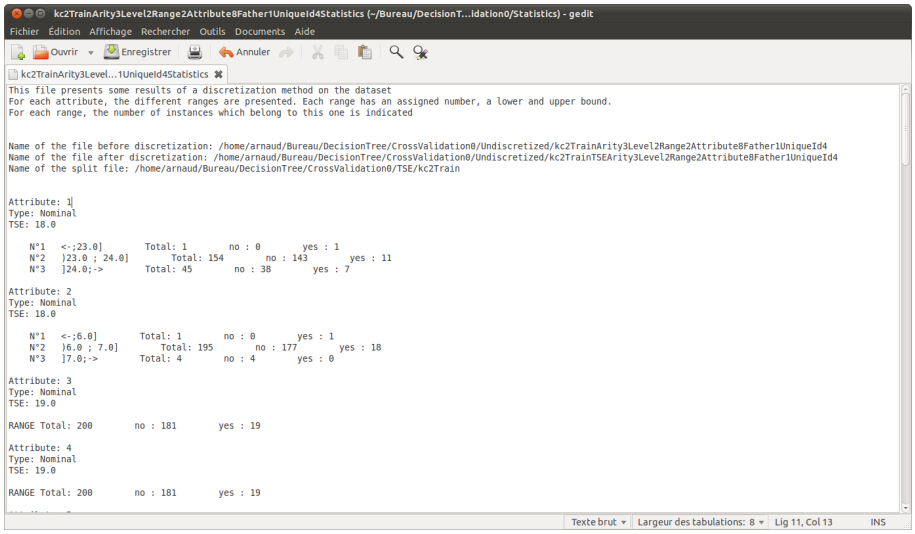

FIGURE A.3 – Aperçu d'un fichier de statistiques pour un noeud

A.1.2.2 Règles

Le logiciel implémenté permet de donner une autre vue de l'arbre de décision, une vue sous la forme d'un ensemble de règles. La Figure A.4 donne un aperçu de la fenêtre de visualisation des règles. Chaque règle a un identifiant, une longueur (qui correspond au nombre conditions de la partie antécédente), la valeur de la partie conséquente, la distribution de celle-ci et la règle complète.

A.1.2.3 Application

Par défaut, le logiciel utilise la validation croisée. Néanmoins, une option est disponible afin d'appliquer un arbre de décision à un jeu de données de test.

A.1.3 Evolution

Un des objectifs de ce mémoire était également de créer un logiciel qui peut servir de base aux chercheurs de l'université Pablo de Olavide [1]. A partir de celui-ci, les chercheurs peuvent faire de nombreuses expérimentations, peuvent implémenter de nouvelles segmentations, de nouvelles mesures, de nouvelles visualisations, de nouvelles fonctionnalités et faire ainsi évoluer le logiciel.

1. Séville - Espagne

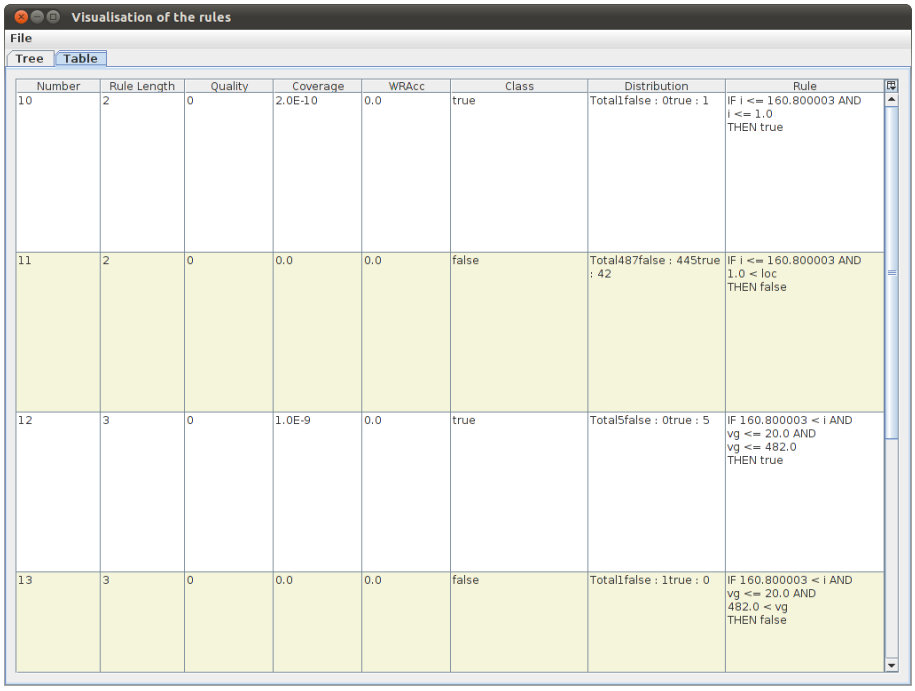

FIGURE A.4 – Fenêtre permettant de visualiser les règles produites

A.2 Taxonomie des méthodes de discrétisation

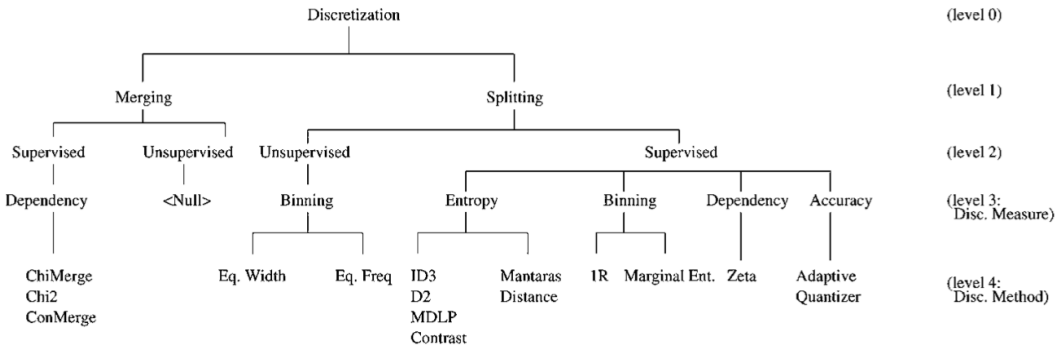

FIGURE A.5 – Différentes méthodes de discrétisation [min]

Method	Taxonomy (corresponding to Section 2)										
	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Equal-width Equal-frequency Fixed-frequency	primary	unsupervised	parametric	non-hierarchical	univariate	disjoint	global	eager	time-insensitive	nominal	non-fuzzy
Multi-interval-entropy-minimization	primary	supervised	non-parametric	hierarchical	univariate	disjoint	global	eager	time-insensitive	nominal	non-fuzzy
ChiMerge StatDisc InfoMerge	primary	supervised	non-parametric	hierarchical	univariate	disjoint	global	eager	time-insensitive	nominal	non-fuzzy
Cluster-based	primary	unsupervised	non-parametric	hierarchical	multivariate	disjoint	global	eager	time-insensitive	nominal	non-fuzzy
ID3	primary	supervised	parametric	hierarchical	univariate	disjoint	local	eager	time-insensitive	nominal	non-fuzzy
Non-disjoint	composite	unsupervised	*	non-hierarchical	univariate	non-disjoint	global	eager	time-insensitive	nominal	non-fuzzy
Lazy	composite	*	*	*	univariate	non-disjoint	global	lazy	time-insensitive	nominal	non-fuzzy
Dynamic-qualitative	primary	unsupervised	non-parametric	non-hierarchical	univariate	disjoint	local	lazy	time-sensitive	nominal	non-fuzzy
Ordinal	composite	*	*	*	univariate	disjoint	global	eager	time-insensitive	ordinal	non-fuzzy
Fuzzy	composite	*	*	*	univariate	non-disjoint	global	eager	time-insensitive	nominal	fuzzy
Iterative-improvement	composite	supervised	*	hierarchical	multivariate	disjoint	global	eager	time-insensitive	nominal	non-fuzzy

Note: each entry of the taxonomy is
0. primary *vs.* composite;
1. supervised *vs.* unsupervised;
2. parametric *vs.* non-parametric;
3. hierarchical *vs.* non-hierarchical;
4. univariate *vs.* multivariate;
5. disjoint *vs.* non-disjoint;
6. global *vs.* local;
7. eager *vs.* lazy;
8. time-sensitive *vs.* time-insensitive;
9. ordinal *vs.* nominal;
10. fuzzy *vs.* non-fuzzy.
An entry filled with '*' indicates that the corresponding method can be conducted in either way of the corresponding taxonomy entry.
This often happens for composite methods, whose taxonomy depends on their primary methods.

FIGURE A.6 – Taxonomie plus complète [YWW10]

A.3 Compléments sur les jeux de données utilisés

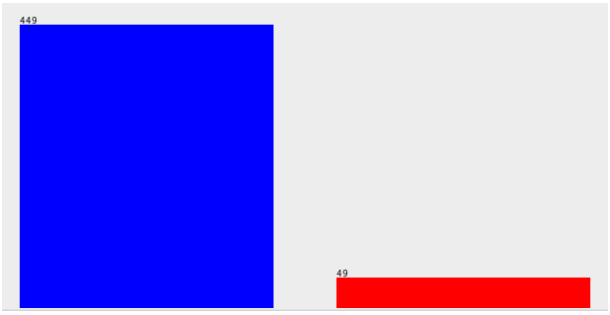

FIGURE A.7 – Distribution CM1

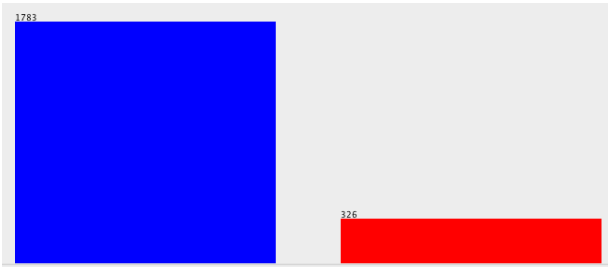

FIGURE A.8 – Distribution KC1

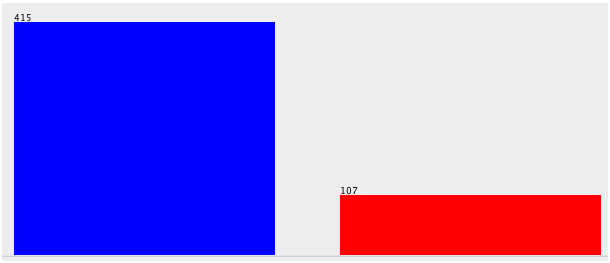

FIGURE A.9 – Distribution KC2

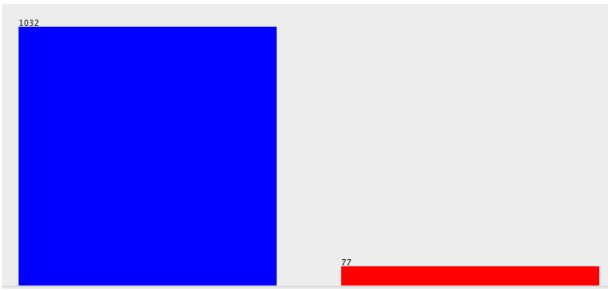

FIGURE A.10 – Distribution PC1

Bibliographie

[Arb] Tutoriel sur les arbres de décision. `http://decisiontrees.net/`. (consulté le 11/10/2010).

[Az0] Jérôme Azé. Entropie. `www.lri.fr/~aze/enseignements/ifips/apprentissage/docs/cours-entropie.pdf`, 2009,2010. (consulté le 09/01/2011).

[BFOS84] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.

[Bre96] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24 :123–140, August 1996.

[Buh] Joachim Buhmann. Dynamic programming. `http://www.inf.ethz.ch/personal/vroth/wrk/stma4_6_volker.pdf`.

[BV98] Tom Brijs and Koen Vanhoof. Cost sensitive discretization of numeric attributes. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, PKDD '98, pages 102–110, London, UK, 1998. Springer-Verlag.

[BW08] Sara Boslaugh and Paul Andrew Watters. *Statistics in a Nutshell*. O'Reilly, 2008.

[Car10] François Caron. Apprentissage automatique : Arbres de décision. `http://www.math.u-bordeaux1.fr/~fcaron/lectures/fall2010/docs/lecture5_pres.pdf`, 2010. (consulté le 03/12/2010).

[CBHK02] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE : Synthetic Minority Oversampling Technique. *Journal of Artificial Intelligence Research*, 16 :321–357, 2002.

[cla] `www.aw-bc.com/info/kumar/assets/downloads/ch4_lecture_notes.pdf`. (consulté le 01/11/2011).

[Coh95] William W. Cohen. Fast Effective Rule Induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[CS98] Philip K. Chan and Salvatore J. Stolfo. Toward scalable learning with non-uniform class and cost distributions : A case study in credit card fraud detection. In *In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 164–168. AAAI Press, 1998.

[data] `nat.54.free.fr/M2_ACSI/Datamining/Datamining.ppt`. (consulté le 29/09/2010).

[datb] `http://lsirwww.epfl.ch/courses/dis/2007ws/lecture/week\%2013\%20Datamining-Association\%20rules.pdf`. (consulté le 16/09/2010).

[datc] Uci repository. `http://archive.ics.uci.edu/ml/`. (consulté le 26/09/2010).

[Dis] Introduction sur la discrétisation. `http://www.info.univ-angers.fr/~gh/wstat/discr.php/`. (consulté le 24/09/2010).

[DSC] Ngoh Woon Hiong Darryl Seet and Tan Ting Chuen. Supervised and unsupervised discretization of continuous features.

[DWK05] Dursun Delen, Glenn Walker, and Amit Kadam. Predicting breast cancer survivability : a comparison of three data mining methods. *Artif. Intell. Med.*, 34 :113–127, June 2005.

[Elk01] Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 2*, pages 973–978, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[ER02] Tapio Elomaa and Juho Rousu. Fast minimum training error discretization. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 131–138, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[Faw06] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27 :861–874, June 2006.

[Fee] A.J. Feelders. Classification : Basic concepts, decision trees and model evaluation. `http://www.cs.uu.nl/docs/vakken/dm/dmhc5.pdf`. (consulté le 18/11/2010).

[FI93] Fayyad and Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. pages 1022–1027, 1993.

[F.I10] Barbara F.I.Pieters. Subgroup discovery on numeric and ordinal targets, with an application to biological data aggregation. Technical Report UU-CS-2010-012, Department of Information and Computing Sciences, Utrecht University, 2010.

[GB10] Jerzy W. Grzymala-Busse. Rule induction. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 249–265. Springer US, 2010. 10.1007/978-0-387-09823-$4_1$3.

[Gen09] Mathieu Gentes. Cours de probabilités et statistiques. `http://www.ann.jussieu.fr/gentes/documents/cours2.pdf`, 2009.

[GLK03] Dragan Gamberger, Nada Lavrac, and Goran Krstacic. Active subgroup mining : a case study in coronary heart disease risk group detection. *Artificial Intelligence in Medicine*, 28(1) :27 – 57, 2003.

[Gom07] Alan Keller Gomes. Small disjuncts grouping by rule coverage and accuracy measures. In *Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications*, pages 412–415, Washington, DC, USA, 2007. IEEE Computer Society.

[HAP89] Robert Holte, Liane Acker, and Bruce Porter. Concept learning and the problem of small disjuncts. Technical report, Austin, TX, USA, 1989.

[HCGdJ10] Franciso Herrera, Cristóbal Carmona, Pedro González, and María del Jesus. An overview on subgroup discovery : foundations and applications. *Knowledge and Information Systems*, pages 1–31, 2010.

[HMS66] E. B. Hunt, J. Marin, and P. J. Stone. *Experiments in induction.* Academic Press, New York, NY, 1966.

[hum00] Data mining and the human genome. `http://www.fas.org/irp/agency/dod/jason/genome.pdf`, 2000. (consulté le 27/03/2011).

[ide] Chapter : Learning by building identification trees. Artificial Intelligence.

[int] Introdution à l'intelligence artificielle : l'apprentissage inductif. (consulté le 17/10/2010).

[JBVW06] Davy Janssens, Tom Brijs, Koen Vanhoof, and Geert Wets. Evaluating the performance of cost-based discretization versus entropy-and error-based discretization. *Comput. Oper. Res.*, 33 :3107–3123, November 2006.

[JF08] Frederik Janssen and Johannes Fürnkranz. An empirical investigation of the trade-off between consistency and coverage in rule learning heuristics. In *Proceedings of the 11th International Conference on Discovery Science*, DS '08, pages 40–51, Berlin, Heidelberg, 2008. Springer-Verlag.

[JF10] Frederik Janssen and Johannes F^{o}rnkranz. On the quest for optimal rule learning heuristics. *Mach. Learn.*, 78 :343–379, March 2010.

[Jin07] Ruoming Jin. Classification : Basic concepts and decision trees. `http://www.cs.kent.edu/~jin/DM07/ClassificationDecisionTree.ppt`, 2007. (consulté le 26/10/2010).

[JMG95] Nathalie Japkowicz, Catherine Myers, and Mark Gluck. A novelty detection approach to classification. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, pages 518–523, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[KHM97] Miroslav Kubat, Robert Holte, and Stan Matwin. Learning when negative examples abound. In *Proceedings of the 9th European Conference on Machine Learning*, pages 146–153, London, UK, 1997. Springer-Verlag.

[KHM98] Miroslav Kubat, Robert C. Holte, and Stan Matwin. Machine learning for the detection of oil spills in satellite radar images. *Mach. Learn.*, 30 :195–215, February 1998.

[KLZ05] Petra Kralj, Nada Lavrac, and Blaz Zupan. Subgroup visualization. In *Proceedings of the 8th International Multiconference Information Society (IS 2005)*, pages 228–231, 2005.

[LCCC10] Wei Liu, Sanjay Chawla, David A. Cieslak, and Nitesh V. Chawla. A Robust Decision Tree Algorithm for Imbalanced Data Sets. In *SDM*, pages 766–777. SIAM, 2010.

[LCGF04] Nada Lavrac, Bojan Cestnik, Dragan Gamberger, and Peter Flach. Decision support through subgroup discovery : Three case studies and the lessons learned. *Mach. Learn.*, 57 :115–143, October 2004.

[LHTD02] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization : An enabling technique. *Data Min. Knowl. Discov.*, 6 :393–423, October 2002.

[LKFT04] Nada Lavrač, Branko Kavšek, Peter Flach, and Ljupčo Todorovski. Subgroup discovery with cn2-sd. *J. Mach. Learn. Res.*, 5 :153–188, December 2004.

[LLV07] Stéphane Lallich, Philippe Lenca, and B Vaillant. Construction of an off-centered entropy for supervised learning. In *XIIth International Symposium on Applied Stochastic Models and Data Analysis (AMSDA 07), Chania, Crete, Greece*, 2007.

[Mar] Simon Marcellin. Arbres de décision en situation d'asymétrie. (consulté le 09/04/2011).

[MIIB] Barbara Mucha, Tania Irani, Irem Incekoy, and Mikhail Bautin. Association rule mining. http://www.cs.sunysb.edu\%2F~cse634\%2Fpresentations\%2FCSE634-Association\%2520and\%2520Web\%2520Mining\%2520Group\%25206.ppt\&ei=zF3eTY21NMHtOdbFlewJ\&usg=AFQjCNEYmrd0SADBrchaYK6h0DwANcHDNQ. (consulté le 20/02/2011).

[min] Data preprocessing : Data reduction-discretization. math.uprm.edu/~edgar/dm8.ppt. (consulté le 15/10/2010).

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[MJPS94] Kamal Ali Michael J Pazzani, Patrick Murphy and David Schulenburg. Trading off coverage for accuracy in forecasts : Applications to clinical data analysis. 1994.

[MR05] Oded Maimon and Lior Rokach. Introduction to supervised methods. In *The Data Mining and Knowledge Discovery Handbook*, pages 149–164. 2005.

[NF] Monique Noirhomme-Fraiture. Cours data mining.

[NLW09] Petra Kralj Novak, Nada Lavrač, and Geoffrey I. Webb. Supervised descriptive rule discovery : A unifying survey of contrast set, emerging pattern and subgroup mining. *J. Mach. Learn. Res.*, 10 :377–403, June 2009.

[Ora] Documentation orange. http://orange.biolab.si/. (consulté le 30/10/2010).

[PT98] Petra Perner and Sascha Trautzsch. Multi-interval discretization methods for decision tree learning. In *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, SSPR '98/SPR '98, pages 475–482, London, UK, 1998. Springer-Verlag.

[Qui93] J. Ross Quinlan. *C4.5 : programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[RCRAR09] Daniel Rodriguez, Riquel Jesus C., Roberto Ruiz, and Jesus S. Aguilar-Ruiz. Searching for rules to find defective modules in unbalanced data sets. In *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering*, SSBSE '09, pages 89–92, Washington, DC, USA, 2009. IEEE Computer Society.

[reg] http://databases.about.com/od/datamining/a/datamining.htm. (consulté le 06/02/2011).

[rep] Répertoire contenant les jeux de données utilisés dans le cadre des expérimentations de notre solution. http://promise.site.uottawa.ca/SERepository.

[RM05] Lior Rokach and Oded Maimon. Decision trees. In *The Data Mining and Knowledge Discovery Handbook*, pages 165–192. 2005.

[Rou01] Juho Rousu. *Efficient Range Partitioning in Classification Learning.* PhD thesis, Department of Computer Science, University of Helsinki, 2001.

[RSE94] Patricia Riddle, Richard Segal, and Oren Etzioni. Representation design and brute-force induction in a boeing manufacturing domain. *Applied Artificial Intelligence*, 8 :125–147, 1994.

[RTL09] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. In *Encyclopedia of Database Systems*, pages 532–538. 2009.

[Ste08] Jerzy Stefanowski. Data mining for imbalanced data : Improving classifiers by selective pre-processing of examples. 2008.

[SZDC95] F Schoonjans, A Zalata, C E Depuydt, and F H Comhaire. Medcalc : a new computer program for medical statistics. *Comput Methods Programs Biomed*, 48(3) :257–62, 1995.

[tes] Test statistique - principe. `http://archive.ics.uci.edu/ml/`.

[TFL00] Ljupco Todorovski, Peter A. Flach, and Nada Lavrac. Predictive performance of weghted relative accuracy. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 255–264, London, UK, 2000. Springer-Verlag.

[TNPLL08] Do Than-Nghi, Nguyen-Khang Pham, Stéphane Lallich, and Philippe Lenca. Expérimentation de l'entropie décentrée pour le traitement des classes déséquilibrées en induction par arbres. In *4e atelier de qualité des données et des connaissances - 8es journées d'extraction et de gestion des connaissances, EGC'08*, pages 39–49, Sophia-Antipolis, France, 2008. Springer-Verlag.

[Tur05] Nicolas Turenne. Arbre de régression et de classification. `genome.jouy.inra.fr/~turenne/cours/coursDM_CART.pdf`, 2005. (consulté le 18/01/2011).

[VC06] Miha Vuk and Tomaž Curk. Roc curve, lift chart and calibration plot. *Metodološki zvezki*, 3(1) :89–108, 2006.

[Wei99] Gary M. Weiss. Timeweaver : a genetic algorithm for identifying predictive patterns in sequences of events. In *In Proceedings of the Genetic and Evolutionary Computation Conference*, pages 718–725. Morgan Kaufmann, 1999.

[Wei04] Gary M. Weiss. Mining with rarity : a unifying framework. *SIGKDD Explor. Newsl.*, 6 :7–19, June 2004.

[Wei05] Gary M. Weiss. Mining with rare cases. In *The Data Mining and Knowledge Discovery Handbook*, pages 765–776. 2005.

[Wek] Documentation weka. `http://www.cs.waikato.ac.nz/ml/weka/`. (consulté le 23/10/2010).

[WFH11] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining : Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3. edition, 2011.

[WH98] Gary M. Weiss and Haym Hirsh. Learning to predict rare events in event sequences. In *In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 359–363. AAAI Press, 1998.

[WP03] Gary M. Weiss and Foster Provost. Learning when training data are costly : the effect of class distribution on tree induction. *J. Artif. Int. Res.*, 19 :315–354, October 2003.

[YWW10] Ying Yang, Geoffrey I. Webb, and Xindong Wu. Discretization methods. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 101–116. Springer US, 2010. 10.1007/978-0-387-09823-4_6.